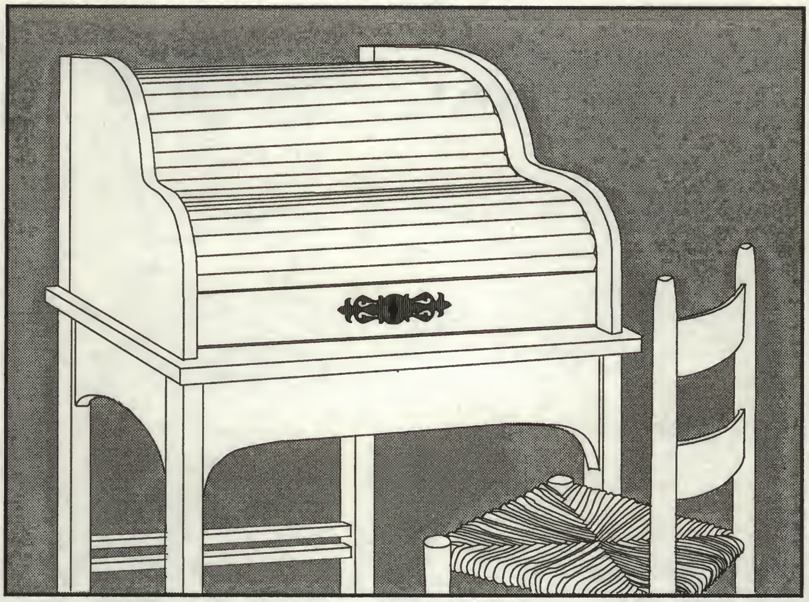
SCO UNIX® System V/386

Operating System

Tutorial

SCO Document Number: 5-22-89-6.0/3.2.0C

Contents



Chapter 1

Introduction

Introduction

This tutorial is an introduction to the use of your UNIX system. It is intended for users who have little or no familiarity with UNIX systems.

The operating system is a software package that controls the actions of your computer system. It makes it easy for you, as a user, to get the computer to do some very complex tasks.

For example, if you want to find out who is currently using the system, just type the command **who**. The operating system calls up an already-existing program that tells the computer to find out who is logged in, and to display the list of names on your screen. If you had to use a programming language to tell the computer to display the list, it would take several lines of code. With UNIX operating systems, there is no need to learn a programming language, because the programs have already been written for you. You never actually see these programs; what you see are the results of executing them when you type the one-word commands.

Another feature of UNIX systems is that they allow more than one person at a time to use the computer system. It does this by taking advantage of the speed with which computers operate. The operating system stores all of the commands from every user and gives them to the computer's hardware one at a time. The operating system and the hardware work so quickly that each user perceives his or her command as being executed immediately. In fact, you will probably not even be able to tell that anyone else is using the system.

In addition to allowing more than one person to use the system at the same time, UNIX systems also permit the simultaneous running of various printers, other peripherals, and tasks. For these reasons, UNIX systems are referred to as *multi-user*, *multi-tasking* operating systems.

The aim of this tutorial is to teach you how to do useful work on a UNIX system as quickly as possible. UNIX systems are distributed with over two hundred commands and programs. The commands and programs described in this tutorial are those that you will use most often, and those that you will find most useful. To this end, it is not necessary to provide you with complete information about each command described in this tutorial. For complete information, refer to the appropriate sections of the *User's Reference* and the *User's Guide*.

1

About This Tutorial

This tutorial is organized as follows:

- Chapter 1, ''Introduction,'' presents an overview of the contents of the entire tutorial, and explains how to use it.

- Chapter 2, ''Basic Concepts,'' explains the concepts that you need to understand to work effectively in the UNIX environment. The chapters that follow presuppose an understanding of the material presented in this chapter.

- Chapter 3, ''Logging In,'' explains how to log in to the system, how to keep your account secure, how to edit the login prompt and how to enter UNIX commands.

- Chapter 4, ''Working with Files and Directories,'' explains how to perform some of the basic tasks involving files and directories. This chapter explains how to create files and directories, how to move, copy, delete, and rename files and directories. The chapter also explains how to use various UNIX text processing utilities, and how to use access permissions with files and directories.

- Chapter 5, ''Housekeeping,'' explains how to use UNIX ''housekeeping'' utilities. This chapter explains how to create backups, how to copy diskettes, how to get information about the status of the system, and how to place commands in the background. The chapter also contains a brief discussion of shell programming.

- Chapter 6, ''UNIX Desktop Utilities,'' explains how to use the UNIX desktop utilities. This chapter explains how to use the automatic reminder service, how to communicate with other users on the system and how to use the system calculator.

The best way to use this tutorial is to begin by reading Chapter 2. This will provide you with the background information that you need in order to understand the material presented in subsequent chapters. You should then read Chapters 3 through 6 at your terminal, entering commands as instructed in the examples.

Each section of each chapter is a self-contained unit. You do not have to read previous sections in order to understand the material presented in any particular section. If you only need to know how to perform a specific task, you can turn to the section of the chapter that explains how to perform that task. For example, if you already know how to create files but are not sure how to print them, turn to ''Printing Files'' in Chapter 4, ''Working with Files and Directories.'' In this case, you do not have to read the first sections of Chapter 4 in order to understand ''Printing Files.''

Chapters 2 through 6 contain a summary of the material covered in each of those chapters.

Notational Conventions

This tutorial uses the following notational conventions:

- Examples in the text are indented.
- *Directories* and *filenames* are printed in *italics*.
- *New concepts* reviewed in the chapter summaries are printed in *italics*.
- **Commands** that you enter are printed in **boldface** type.
- **Keys** to be pressed are printed in **boldface** type. For example, the Return key is represented by:

  ⟨Return⟩

- **Key combinations** are printed in **boldface** and are hyphenated. An example is:

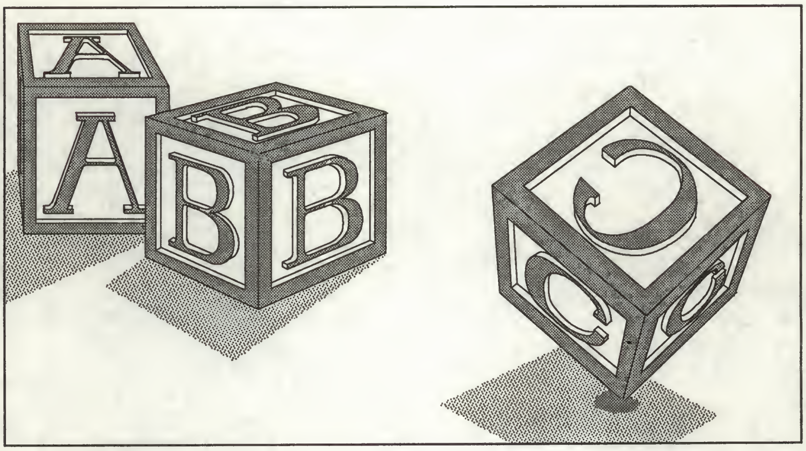
  ⟨CTL⟩d

  When you see a key combination, you are supposed to hold down the first key and press the second key. In this example, you should hold down the Control key and press the d key.

- An uppercase letter in parentheses is often appended to command names, as in:

  touch(C)

  The letter in parentheses refers to the section of the *User's Reference* that contains complete information on the command.



Chapter 2

Basic Concepts

Introduction

This chapter explains the basic concepts that you need to understand to work effectively in the UNIX environment. After reading this chapter, you should understand the fundamentals of user accounts, as well as how the system's files and directories are organized and named, how commands are entered, and how a command's input and output can be redirected. It is important to read this chapter before proceeding to the tutorial chapters that follow.

2

Accounts

To organize and record each user's activities, the system administrator gives everyone an account. There are two main types of accounts: User and Super User. Both of these are described in the following sections.

User Accounts

User accounts are the type most commonly issued. They are given to anyone who needs to log on to a UNIX system. Your user account contains the following information:

- Your login name. This is the name by which you are known on the system. It is the name you enter at the login prompt.

- Your password. To increase system security, each user may be given a password. This password is entered when you log on to the system.

- Your group identification. Each user is known to the system as an individual and as a member of a group. Group membership is important for system security reasons. As a member of a group, you may be permitted to access files and directories that you cannot access as an individual.

- Your ''home directory.'' This is the place in the filesystem where you can keep personal files. When you first log in to the system, you are placed in your home directory.

- Your ''login shell.'' This is the program that reads and executes the UNIX commands you input. In most cases, your login shell will be the ''Bourne shell.'' The Bourne shell uses the dollar sign (**$**) as a prompt. However, you may be configured to use the ''C-shell,'' which uses the percent sign (**%**) as a prompt. Throughout this tutorial, the expression ''UNIX prompt'' is used to refer to your shell prompt, whether it is the percent sign or the dollar sign.

Once an account has been established for you, you can manipulate the files, directories, and commands that make up a UNIX system.

Super User Account

In addition to each user's individual account, every UNIX system has a ''super user'' account. (The super user is also referred to as ''root.'') In order to perform certain system administration tasks, the system administrator must log in as the super user. The super user has free rein over the system. The super user can read and edit any file on the system, as well as execute any program.

2

Files

The file is the fundamental unit of the UNIX filesystem. There are three different types of UNIX files: ordinary files (what we usually mean when we say ''file''), special device files, and directories. Each of these is described in the sections that follow.

Ordinary Files

An ordinary file is simply a collection of 8-bit bytes. Ordinary files are usually documents, program source code, or program data. Executable binary files, or computer programs, are also considered ordinary files. The bytes of an ordinary file are interpreted as text characters, binary instructions, or program statements, by the programs that examine them.

Every ordinary file has the following attributes:

- a filename (not necessarily unique),
- a unique filesystem number called an inode number,
- a size in bytes,
- a time of last change,
- a set of access permissions,
- an owner and a group.

File Protection

On a multi-user system, it is often necessary to ''protect'' certain files, denying some users access to the files while allowing access to others. Files are protected by assigning appropriate ''access permissions'' to them. UNIX systems provide three levels of access permissions:

read Having read permission on a file allows a user to view the contents of the file with such commands as **cat** and **more**. A user with read-only permission cannot edit a file.

write	Having write permission on a file allows a user to edit the file.
execute	If the file is a program, having execute permission on the file allows a user to run the program. You cannot run a program for which you do not have execute permission.

Access permissions are assigned by a file's owner. (By default, the owner of a file is its creator.) Any combination of the three levels is permitted. This allows the file's owner to determine which users can read, write, and/or execute the file. Note that the super user has read, write, and execute permissions on all files on the system.

The UNIX file security mechanism is very flexible. It allows separate access permissions to be set for a file's owner, a file's group, and for all other users. In a typical case, the owner of a file might have read and write permissions, the group read-only permission, and all other users no access permissions at all.

Special Device Files

Each physical device on the system, such as hard and floppy disks, lineprinters, terminals, and system memory, is assigned to a "special file." These files are also called "special device files." Special device files are not discussed in this tutorial. (For more information on special device files, see the *System Administrator's Guide*.)

Directory Files

Directory files are more like file drawers than files. They are places where files are stored (conceptually, not physically). A directory file is usually referred to as a "directory," and contains the names and locations of the files "within it."

Like ordinary files, directories can be protected by assigning appropriate access permissions. These are read, write and execute. In order to do anything useful in a directory, a user must have execute permission on that directory. Execute and write permissions determine whether files can be added to or removed from a directory. Execute and read permissions determine whether the contents of a directory can be listed. Access permissions are assigned to a directory by its owner. By default, the owner of a directory is its creator.

Directory Structure

With multiple users working on multiple projects, the number of files in a filesystem can proliferate rapidly, creating an organizational nightmare. The inverted "tree-structured" directory hierarchy that is a feature of UNIX systems allow users to organize large numbers of files efficiently. Related files can be grouped together in a single directory. In addition to ordinary files, a directory can contain other directories, sometimes called "subdirectories." Subdirectories themselves can contain ordinary files and more subdirectories, and so on. The **cd** command is used to move from one directory to another.

In this typical tree of files, the root of the tree is at the top and the branches of the tree grow downward. Directories correspond to "nodes" in the tree, while ordinary files correspond to "leaves." Figure 2-1 represents this inverted tree-structured directory hierarchy.

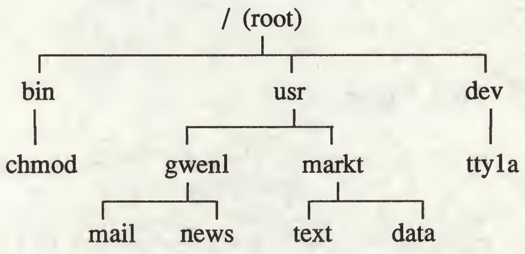


Figure 2-1 A Typical Filesystem

In Figure 2-1, the names *bin*, *usr*, *dev*, *gwenl*, and *markt* all represent directories, and are all nodes in the tree. At the top of the tree is the root directory, which is given the name slash (/). The names *mail*, *news*, *text*, and *data* all represent ordinary files, and they are all "leaves" of the tree. The file *chmod* is the name of a command that can be executed. The name *tty1a* is a special device file. It represents a terminal and is also represented in the tree.

If a directory contains a downward branch to other files or directories, those files and directories are "contained" in the given directory. All directories and files on the system are contained in the root directory. In Figure 2-1, the files *mail* and *news* are contained in the directory *gwenl*, which itself is contained in the directory *usr*. The directory *usr*, in turn, is contained in the root directory.

It is possible to name any file in the system by starting at the root and traveling down any of the branches to the desired file. Files can also be named relative to any directory. UNIX naming conventions are discussed later in this chapter.

The User Directory

Each user is given a personal or ''home'' directory. This is a place where you can keep files that no other user is likely to need. Within the home directory, you may have other subdirectories that you own and control. All of the home directories on a UNIX system are often placed in the *usr* directory, as illustrated by Figure 2-2.

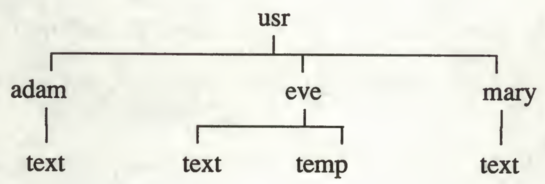


Figure 2-2 A Typical User Directory

In Figure 2-2, the *usr* directory contains each user's home directory. There are three users on this system, *adam, eve*, and *mary*.

Naming Conventions

Every single file, directory, and device on a UNIX system has both a filename and a pathname. Filenames and pathnames are discussed in the following two sections.

Filenames

A filename is a sequence of 1 to 14 characters consisting of letters, digits and other special characters such as the underbar (_). Every single file, directory, and device in the system has a filename. Although you can use almost any character in a filename, it is best to confine filenames to the alphanumeric characters and the period. Other characters, especially control characters, are discouraged for use in filenames.

Filenames should be indicative of a file's contents. For example, a file containing purchase orders should have a name like *orders*, rather than *file1*. Note that filenames must be unique only within directories and need not be unique system-wide. Different directories can contain different files that have the same name. For example, there can be several files named *text* on a single system, as long as those files are each in separate directories.

When a filename contains an initial period, it is "hidden," and it is not displayed by the **lc** command. System configuration files are often hidden. However the **lc -a** command does display hidden files. The dash (-) is used in specifying command options and should be avoided when naming files. In addition, the question mark (?), the asterisk (*), brackets ([and]), and all quotation marks should *never* be used in filenames, because they have special meaning to the UNIX shell. (For more information on these characters, see "Special Characters" later in this chapter.)

Pathnames

A pathname is a sequence of directory names followed by a simple filename, each separated from the previous name by a slash. If a pathname begins with a slash, it specifies a file that can be found by beginning a search at the *root* of the entire tree. Otherwise, files are found by beginning the search at the user's *current directory* (also known as the *working directory*). The **pwd** command is used to print the name of the working directory on the screen.

A pathname beginning with a slash is called a *full* or *absolute pathname*. The absolute pathname is a map of a file's location in the system. Absolute pathnames are unique: no two files, directories, or devices have the exact same absolute pathname. A pathname *not* beginning with a slash is called a *relative pathname*, because it specifies a path relative to the current directory.

Sample Names

Among the directory and file names commonly found on UNIX systems are:

/	The name of the root directory.
/bin	The directory containing most of the frequently used UNIX commands.
/usr	The directory containing each user's personal directory. The subdirectory, */usr/bin* contains frequently used UNIX commands not in */bin*.
/dev	The directory containing special device files.
/dev/console	The special device file associated with the system console.
/dev/tty*XX*	The names of special device files associated with system ports. *XX* represents a number, such as *1a* or *006*. Most ports are assigned to terminals.
/lib	The directory containing files of "libraries" used for system development.
/usr/lib	The directory containing directories with UNIX applications.
/tmp	The directory for temporary files.
/usr/joe/run	A typical full pathname. It is the pathname of a file named *run* belonging to a user named *joe*.

bin/script	A relative pathname. It names the file *script* in subdirectory *bin* of the current working directory. If the current directory is the root directory (/), it names */bin/script*. If the current directory is */usr/joe*, it names */usr/joe/bin/script*.
file1	Name of an ordinary file in the current directory.

All files and directories, with the exception of the root directory, have a ''parent'' directory. This directory is located immediately above the given file or directory. The UNIX filesystem provides special shorthand notations for the parent directory and for the current directory:

.	The shorthand name of the current directory. For example, *./filexxx* names the same file as *filexxx,* in the current directory.
..	The shorthand name of the current directory's parent directory. For example, the shorthand name *../..* refers to the directory that is two levels ''above'' the current directory.

Special Characters

UNIX systems include a facility for specifying sets of filenames that match particular patterns. Suppose, for example, you are working on a large book. The different chapters of the book might be kept in separate files, whose names might be *chpt1, chpt2, chpt3*, and so on. You might even break each chapter into separate files. For example, you might have files named *chpt1.1, chpt1.2, chpt1.3*, and so on.

If you want to print the whole book on the lineprinter, you could enter the following command:

```
lp chap1.1 chap1.2 chap1.3 ...
```

Entering so many filenames in a command quickly becomes tedious, and will probably lead to mistakes. Fortunately, there is a shortcut. A sequence of names containing a common pattern can be specified with the use of special ''wildcard'' characters. The wildcard characters discussed in this chapter are:

*	Matches zero or more characters
[]	Matches any character inside the brackets
?	Matches any single character

For example, you can enter:

```
lp chap*
```

The asterisk (*) means ''zero or more characters of any type,'' so this command translates into ''send all files whose names begin with the word *chap* to the lineprinter.'' This is a quick and effective way of printing all the files that make up your book.

This shorthand notation is not a unique property of the **lp** command. It can be used with any command. For example, you can list the names of the files in the book by typing:

```
lc chap*
```

The asterisk is not limited to the last position in a filename. It can be used anywhere in a filename and can occur several times. An asterisk by itself matches all filenames not containing slashes or beginning with periods:

```
cat *
```

This command displays all files in the current directory on your terminal screen.

The asterisk is not the only pattern-matching feature available. Suppose you want to print only chapters 1 through 4, and 9. You can enter:

```
lp chap[12349]*
```

The brackets ([and]) mean ''match any of the characters inside the brackets.'' A range of consecutive letters or digits can be abbreviated, so you can also do this with the following command:

```
lp chap[1-49]*
```

(Note that this does *not* try to match *chap1** through *chap49**, but rather *chap1** through *chap4** and *chap9**.) Letters can also be used within brackets: ''[a-z]'' matches any character in the range ''a'' through ''z''.

The question mark (?) matches any single character:

```
lc ?
```

This command lists all files that have single-character names. The following command lists information about the first file of each chapter (i.e., *chap1.1*, *chap2.1*, ...):

```
l chap?.1
```

If you need to turn off the special meaning of any of the wildcard characters (*, ?, and [...]) enclose the entire argument in single quotation marks. For example, the following command lists only a file named "?" rather than all one-character filenames:

```
lc ´?´
```

Pattern-matching features are discussed further in "The Shell" chapter of the *User's Guide*.

Commands

You have already been introduced to three useful UNIX commands, **lc**, **lp**, and **cat**. The **lc** command is used to display directory contents, the **lp** command to print files and the **cat** command to display file contents.

Commands are executable programs. When you enter the name of a command, the Operating System looks for a program with that name and executes the program, if it can be found. Command lines can also contain arguments that specify options or files that the program needs. The command line and command syntax are discussed in the next two sections.

Command Line

The operating system always reads commands from the ''command line.'' The command line is a line of characters that is read by the shell to determine what actions to perform. (There are two shells available: the Bourne shell and the C-shell.) The shell reads the names of commands from the command line, finds the executable program corresponding to the name of the command, then executes that program. When the program finishes executing, the shell resumes reading the command line.

When you enter commands at a terminal, you are actually editing a line of text called the ''command-line buffer.'' The command-line buffer becomes a command line only when you press ⟨Return⟩. The command-line buffer can be edited with the ⟨BKSP⟩ and ⟨CTL⟩u keys. If the **INTERRUPT** key is pressed before ⟨Return⟩, the command-line buffer is erased. (On most keyboards, the ⟨DEL⟩ key is the **INTERRUPT** key.)

Multiple commands can be entered on a single command line, provided they are separated by a semicolon (;). For example, the following command line prints out the current date and the name of the current working directory:

```
date; pwd
```

Commands can be submitted for processing in the ''background'' by appending an ampersand (&) to the command line. This mode of execution is similar to ''batch'' processing on other systems. The main advantage of placing commands in the background is that you can execute other commands from your terminal in the ''foreground'' while the background commands execute. For example, the following command outputs disk usage statistics in the directory */usr*, a fairly time-consuming operation, without tying up your terminal:

```
du /usr > diskuse &
```

The output of this **du** command is placed in the file *diskuse*, by redirecting output with the greater-than symbol <>>. (Redirection of input and output is discussed in ''Input and Output'' below. Background processing is discussed in ''Advanced Tasks.'')

Syntax

The general syntax for commands is:

cmd [*options*][*arguments*] [*filename*] [...]

By convention, command names are lowercase. Options are always preceded by a dash (-) and are not required. They are used to modify the command. For example, the **lc** command lists, in columnar format, the contents of a directory. The same command with the **-l** option (**lc -l**) produces a long listing of a directory's contents.

In some cases, options can be grouped to form a single option argument, as in the following command:

```
lc -rl
```

This command is really a combination of two options, where the **-rl** option selects the option that lists all files in the directory in both reverse alphabetical order and with the long format.

Sometimes multiple options must be given separately, as in the following command:

copy -a -v *source destination*

Here the **-a** option tells the **copy** command to ask the user for confirmation before copying *source* to *destination*. The **-v** option specifies ''verbose'', which causes **copy** to list the names of the files that are copied, as they are copied.

Other arguments, such as search strings, can also be given, as in the following command:

grep *'string of text' data.file*

The *string of text* in this example is a single argument, and is the series of characters, or string, for which the **grep** command searches in the file *data.file*.

2

Input and Output

By default, the operating system assumes that input comes from the terminal keyboard and that output goes to the terminal screen. To illustrate typical command input and output, enter:

cat

This command now expects input from your keyboard. It accepts as many lines of input as you enter, until you press ⟨CTL⟩d which is the "end-of-file" or "end-of-transmission indicator."

For example, enter:

this is two lines ⟨Return⟩
of input ⟨Return⟩
⟨CTL⟩d

The **cat** command immediately outputs each line as you enter it. Since output is sent to the terminal screen by default, that is where the lines are sent. Thus, the complete session will look like this on your terminal screen:

```
$ cat
this is two lines
this is two lines
of input
of input
$
```

The flow of command input and output can be "redirected" so that input comes from a file instead of from the terminal keyboard and output goes to a file or lineprinter, instead of to the terminal screen. In addition, you can create "pipes" to use the output of one command as the input of another. (Redirection and pipes are discussed in the next two subsections.)

Redirection

On UNIX systems, a file can replace the terminal for either input or output. For example, the following command displays a list of files on your terminal screen:

```
lc
```

But if you enter the following command, a list of your files is placed in the file *filelist* (which is created if it does not already exist), rather than sent to the screen:

```
lc > filelist
```

The symbol for output redirection, the greater-than sign (>), means "put the output from the command into the following file, rather than display it on the terminal screen." The following command is another way of using the output redirection mechanism:

```
cat f1 f2 f3 > temp
```

This command places copies of several files in the file *temp* by redirecting the output of **cat** to that file.

The output append symbol (>>) works very much like the output redirection symbol, except that it means "add to the end of." The following command means "concatenate *file1*, *file2*, and *file3* to the end of whatever is already in *temp*, instead of overwriting and destroying the existing contents."

```
cat file1 file2 file3 >> temp
```

As with normal output redirection, if *temp* doesn't already exist, it is created for you.

In a similar way, the input redirection symbol (<) means "take the input for a program from the following file, instead of from the terminal." As an example, you could enter the following command to send a file named *letter.txt* to several people using the UNIX **mail** facility:

```
mail adam eve mary joe < letter.txt
```

(See Chapter 6 of this tutorial for information on **mail**.)

Pipes

One of the major innovations of UNIX systems is the concept of a ''pipe.'' A pipe is simply a way to use the output of one command as the input of another, so that the two run as a sequence of commands called a ''pipeline.''

For example, suppose that you want to find all unique lines in *frank.txt*, *george.txt*, and *hank.txt* and view the result. You could enter the following sequence of commands:

```
sort frank.txt george.txt hank.txt > temp1
uniq < temp1 > temp2
more temp2
rm temp1 temp2
```

But this is more work than is necessary. What you want is to take the output of **sort** and connect it to the input of **uniq**, then take the output of **uniq** and connect it to **more.** You would use the following pipe:

```
sort frank.txt george.txt hank.txt | uniq | more
```

The vertical bar character (|) is used between the **sort** and **uniq** commands to indicate that the output from **sort**, which would normally have been sent to the terminal, is to be used as the input of the **uniq** command, which in turn sends its output to the **more** command for viewing.

The following command is another example of a pipe. The **wc** command counts the number of lines, words, and characters in its input. The **who** command prints a list of users currently logged on, one per line. Thus, the following pipeline tells you the number of users who are logged in by counting the number of lines that come from the **who** command:

```
who | wc -l
```

Notice the difference in output between **wc -l** and **wc**. By default, **wc** tells you how many lines, words, and characters there are in the input. However, **wc -l** tells you only how many lines.

Any program that accepts input from the keyboard can accept input from a pipe instead. Any program that displays output to the terminal screen can send input to a pipe. You can have as many elements in a pipeline as you wish.

Summary

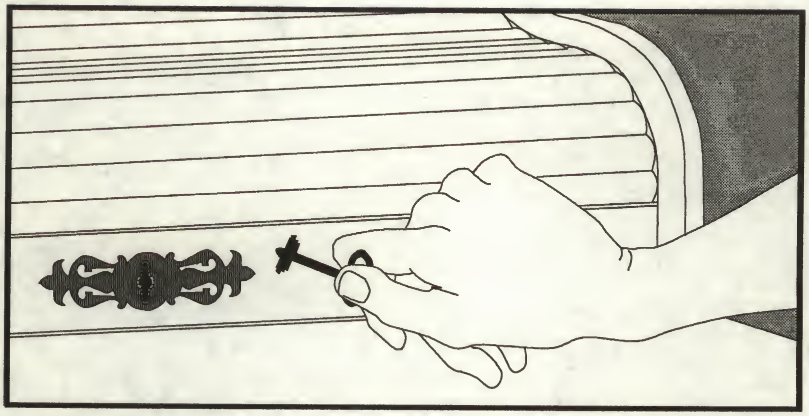
Accounts are given to each user to help organize the computer system and to keep track of everyone's activities. Without an account, you cannot log into the system. There are two types of accounts: *user* and *super user*. User accounts are the more common type, and are given to every user. Super users accounts provide access to all other accounts and files, and are usually only given to the system administrator.

With UNIX systems, all information is stored in *files*. *Special device files* store information about the different hardware components of the system. These pre-made files come with the system, and you cannot manipulate or rename them. *Ordinary files*, on the other hand, can be created, named, and edited by you. Groupings of files are stored in *directories*. Directories can also contain other directories (called *subdirectories*) in addition to files.

You can tell the computer to execute a task by giving it a UNIX *command*. The line on which the command is typed is called the *command line*, and is read by the operating system whenever you press ⟨Return⟩.

You can instruct the operating system to send output to a device other than a terminal screen (such as a printer or a file). Likewise, you can designate an input source to be something other than a terminal. For example, you can use the *pipe* character to tell the operating system to use the output from one command as the input for another. By stringing UNIX commands together in this way, you can create your own customized command sequences.

For a complete explanation of the commands presented in this chapter, see the *User's Guide* and the *User's Reference*.



Chapter 3

Logging In

Introduction

This chapter explains how to perform the following basic tasks on a UNIX system:

- Log in to the system,
- Log out of the system,
- Change your password,
- Reset your terminal type,
- Enter a UNIX command,
- Erase an incorrect command line,
- Stop and start screen output.

This chapter is designed as a tutorial. The best way to use this chapter is to read it at your terminal, entering commands as instructed in the examples.

None of the commands described in this chapter is described in great detail. For a complete explanation of each command, refer to the *User's Reference*.

Gaining Access to the System

To use a UNIX system, you must first gain access to it by logging in. When you log in, you are placed in your home directory. Logging in, changing your password, and logging out are described below.

Logging In

Before you can log into the system, you must be given a system "account." In most cases, your account is created for you by your system administrator. However, if you need to create the account yourself, refer to the *System Administrator's Guide* for information on creating user accounts. This section assumes that your account has already been created.

Normally, the system sits idle and the prompt "login:" appears on the terminal screen. If your screen is blank or displays nonsense characters, press the **INTERRUPT** key a few times. On most keyboards, the **INTERRUPT** key is the ⟨DEL⟩ key.

When the "login:" prompt appears, follow these steps:

1. Enter your login name and press ⟨Return⟩. If you make a mistake as you type, press ⟨CTL⟩u (hold down the ⟨CTL⟩ key and press the **u** key) to start the line again. After you press ⟨Return⟩, "Password:" appears on your screen.

2. Enter your password and press ⟨Return⟩. The letters of your password do not appear on the screen as you enter them, and the cursor does not move. This is to prevent other users from learning your password. If you enter your login name or password incorrectly, the system displays the following message:

   ```
   Login incorrect
   login:
   ```

   If you get this message, enter your login name and password again.

3. Depending on how your system is configured, you may or may not be prompted to enter your terminal type. If you are prompted for your terminal type, you see a line like the following:

```
TERM=(unknown)
```

   Enter your terminal type if you see this line. (If you are not sure how to specify your terminal type, contact your system administrator.)

Once you have entered all the correct information, the ''prompt character'' appears on the screen. This is a dollar sign ($) for Bourne shell users and a percent sign (%) for C-shell users. The prompt tells you that your UNIX system is ready to accept commands from the keyboard.

Depending on how your system is configured, you may also see a ''message of the day'' after you log in.

Logging Out

The simplest way to log out is to enter **logout** at the % prompt for C-shell users, or **exit** at the $ prompt for Bourne shell users. You might also be able to logout by pressing ⟨CTL⟩d at the prompt. However, some systems are configured to prevent logout with ⟨CTL⟩d. The reason for this is that ⟨CTL⟩d signifies the end-of-file on UNIX systems, and it is often used within programs to signal the end of input from the keyboard. Since people sometimes make the mistake of pressing ⟨CTL⟩d several times, they often find themselves unintentionally logged out of the system. To prevent this, system administrators may disable logout with ⟨CTL⟩d.

Familiarize yourself with the logout procedure by pressing ⟨CTL⟩d, if you are currently logged in. If this does not work, log out by entering **logout** (C-shell) or **exit** (Bourne shell). If you are not logged in, log in and then log out, experimenting with ⟨CTL⟩d and with **logout** or **exit**.

Changing Your Password

To prevent unauthorized users from gaining access to the system, each authorized user can be given a password. When you are first given an account on a UNIX system, you are assigned a password by the system administrator. Depending on the security scheme used at your site, you may always be assigned passwords, or, allowed to choose or generate your own. The system can also be set up to check passwords for obviousness. Some UNIX systems require you to change your password at regular intervals. Whether yours does or not, it is a good idea to change your password regularly (at least once every two months) to maintain system security.

Note

> Because system security is completely configurable, the requirements will vary from site to site. Consult the ''Using a Trusted System'' chapter of the *User's Guide* for more information about system security.

If you are permitted to run the **passwd** command, follow these steps to change your password:

1. Enter the following command and press ⟨Return⟩:

   passwd

   You see:

   ```
   Last successful password change for user: Tue Mar 14 10:31:22 1989
   Last unsuccessful password change for username: NEVER

                Choose password

   You can choose whether you pick your own password,
   or have the system create one for you.

          1. Pick your own password
          2. Pronounceable password will be generated for you

   Enter choice (default is 1):
   ```

   Your login name appears in place of *user*. Enter **1** and press ⟨Return⟩.

2. You are then prompted to enter your old password:

   ```
   Old password:
   ```

3. Carefully enter your old password. It is not displayed on the screen. If you make a mistake, press ⟨Return⟩. The message ''Sorry'' appears, then the system prompt. Begin again with step 1.

4. The following message appears after you enter your old password and press ⟨Return⟩:

   ```
   New password:
   ```

   Enter your new password and press ⟨Return⟩. It is generally a good idea to use a combination of numbers and lower-case and upper-case letters in your password.

5. You see the following message:

   ```
   Re-enter new password:
   ```

   Enter your new password again. If you make a mistake, you see the following message:

   ```
   They don't match; try again
   ```

   Begin again with step 1 if you see this message.

When you complete the procedure, the UNIX prompt reappears. The next time you log in, you must enter your new password.

Keeping Your Account Secure

Security is ultimately the responsibility of the user. The careless use and maintenance of passwords represents the greatest threat to the security of a computer system.

Password Security

Here are some specific guidelines for passwords:

1. Don't use passwords that are easy to guess. Passwords should be at least six characters long and include letters, digits, and punctuation marks. (Example: frAiJ6*)

2. Passwords should not be names (even nicknames), proper nouns, or any word found in */usr/dict/words*. (Don't use a password like: terry9)

3. Always keep your password secret. Passwords should never be written down, sent over electronic mail, or verbally communicated. (Treat it like the PIN number for your instant teller card.)

Good Security Habits

There are simple, good security habits. Here are some general guidelines:

1. Remember to log out before leaving a terminal.

2. Use the **lock**(C) utility when you leave your terminal, even for a short time.

3. Make certain that sensitive files are not publically readable. (See the discussion of file and directory permissions in Chapter 4 of this tutorial for information on how to do this.)

4. Keep any floppies or tapes containing confidential data (program source, database backups) under lock and key.

5. If you notice strange files in your directories, or find other evidence that your account has been tampered with, tell your system administrator.

Changing Your Terminal Type

To communicate with the operating system via your terminal, you must tell it what type of terminal you have. On most systems, the system console is already configured for use. However, serial terminals of various types can be connected to a UNIX system. If you are working from a serial terminal, it can be important to know how to specify your terminal type.

The terminal type is displayed each time you log in. You can change the value of the terminal type displayed by editing the *.profile* file in your home directory. If you are using the C-shell, you do not have a *.profile* file. Instead, you must edit the *.login* file in your home directory.

There are at least two reasons why you might want to change the value of the terminal type displayed:

- You might have a new terminal that is not the same model as your old terminal. If so, the terminal type displayed by your old *.profile* (*.login*) file will be incorrect.
- The terminal type displayed might be ''unknown'' or ''ansi'' or another setting which is not correct for your terminal. This would require you to type in your terminal type every time you log in. By changing the terminal type to the setting that is correct for your terminal, all you have to do is press ⟨Return⟩ when prompted for your terminal type. There is no need for you to enter the terminal type.

To permanently change the terminal type displayed, use **vi** to edit *.profile* (*.login*). In order to use **vi** to make these changes, it may be necessary to manually set the terminal type for the current session. To make this initial specification, enter the commands listed below.

Bourne shell:

```
TERM=termtype; export TERM
```

C-shell:

```
setenv TERM termtype
```

where *termtype* is your terminal type.

Once you have temporarily set the terminal type, you can use **vi** to make the appropriate changes in *.profile* (*.login*) so that the the terminal type is set automatically whenever you log in. Chapter 4 of this tutorial explains how to use **vi**.

Once in **vi**, move the cursor to the line that looks like the following:

```
eval 'tset -m :\?unknown -s -r -Q'
```

Change *unknown* (or whatever the value is) in this line to the terminal type of your terminal. For example, if you normally log in on a vt100 terminal, you would change the line to:

```
eval 'tset -m :\?vt100 -s -r -Q'
```

Each time you log in, you would then see the message:

```
TERM = (vt100)
```

Press ⟨Return⟩ and the terminal type is set to vt100. There is no need to enter **vt100**.

Entering Commands

Before you begin working with the commands described in the rest of this tutorial, you should be familiar with three very useful UNIX features. These are character type-ahead and the special key-combinations used to erase the command line, and stop and start screen output. These features are discussed below.

Entering a Command Line

Entering a command line consists of typing characters and then pressing ⟨Return⟩. Once you press ⟨Return⟩, the computer reads the command line and executes the specified commands. No command entered on the command line is executed until ⟨Return⟩ is pressed.

You can enter as many command lines as you want without waiting for the commands to complete their execution and for the prompt to reappear. This is because UNIX systems support character type-ahead. The system can hold up to 256 characters in the kernel buffers that read keyboard input. Experiment with this type-ahead feature by entering the following commands, one right after the other, without waiting for a previous command to finish executing. (Always press ⟨Return⟩ after entering a command. In the following example, press ⟨Return⟩ after entering each command.)

```
lc -la
du -a
lc -Fa
```

These commands generate a long listing of all the files in the current directory, then display disk usage statistics for these files, and finally list the files again, but in a different format.

Erasing a Command Line

Typing errors are bound to occur when you enter commands. To erase the current command line, press ⟨CTL⟩u. When you press ⟨CTL⟩u, a new prompt is displayed and no command is executed.

Halting Screen Output

Data often scrolls across your screen faster than you can read it. To halt scrolling temporarily, press ⟨CTL⟩s. To restart scrolling, press ⟨CTL⟩q. Experiment with ⟨CTL⟩s and ⟨CTL⟩q by entering the following command, then pressing ⟨CTL⟩s to stop the output and ⟨CTL⟩q to restart it:

```
ls /bin
```

Summary

For security reasons, UNIX systems require that all users have a *login name* and a *password*. To access the operating system, enter your login name at the login prompt, and your password at the password prompt. If you enter either incorrectly, you must start over from the beginning.

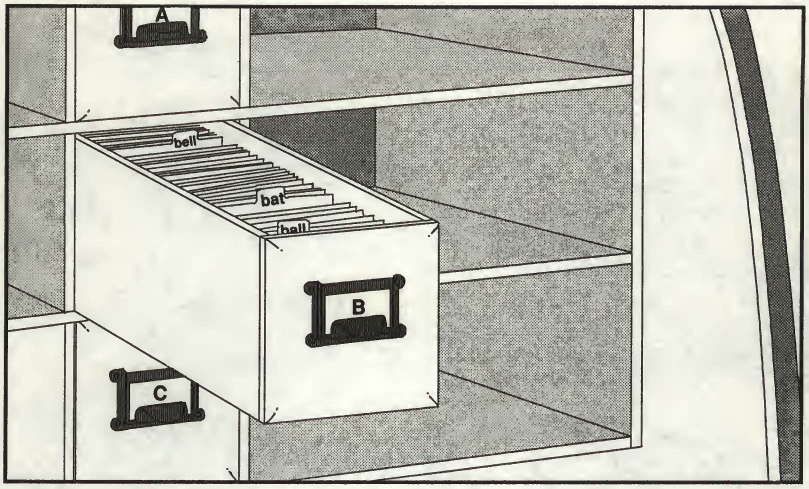
Once you have accessed the system, you can change your password at any time by using the **passwd** command. To enter this or any other UNIX command, you must press ⟨Return⟩ after typing the name of the command on the *command* line. If you need to erase the command line, press ⟨CTL⟩u.

The ⟨CTL⟩s and ⟨CTL⟩q keys respectively stop and continue the scrolling of information across your terminal screen.

If you need to permanently change your terminal type designation, you must edit the file (*.profile* or *.login*) that contains the designation. This file is always in your home directory.

When you are through using the system, enter **logout** if you are using the C-shell, or **exit** if you are using the Bourne shell. After entering this command, you are logged out of the system.

For a complete explanation of the commands presented in this chapter, see the *User's Guide* and the *User's Reference*.



Chapter 4

Working with Files and Directories

Introduction

This chapter explains how to perform the following tasks on a UNIX system:

- Print the name of the current directory,
- List directory contents,
- Change to another directory,
- Create, remove, rename, and copy directories,
- Display the contents of files,
- Delete, combine, rename, move, copy, and search for files,
- Use the full-screen text editor **vi** to create files,
- Print files,
- Compare and sort files,
- Search for patterns in a file,
- Count words, lines and characters in a file,
- Use file and directory permissions.

This chapter is designed as a tutorial. The best way to use this chapter is to read it at your terminal, entering commands as instructed in the examples.

None of the commands described in this chapter is described in great detail. For a complete explanation of each command, refer to the *User's Reference*.

Working with Directories

Because of the hierarchical structure of the UNIX filesystem, any UNIX system has many directories and subdirectories. There are several commands that simplify working in directories. These commands are described in the following sections.

Printing the Name of Your Working Directory

The directory you are "in" at any given time is your "working" directory. All commands are executed relative to the working directory. The name of this directory is given by the **pwd** command, which stands for "print working directory." To find out what your current working directory is, enter the following command:

```
pwd
```

When you first log in to the system, you are placed in your home directory.

Listing Directory Contents

Several related commands are used to list the contents of directories:

lc	This command is a variation of the **ls** command. The **ls** command alphabetizes and displays directory contents. The **lc** command alphabetizes directory contents and displays them in columnar format.
lf	This command does the same as **lc**, and it also marks directories with a slash (/) and executable files (computer programs) with an asterisk (*).

Enter the following command to list the contents of */usr/bin*:

```
lc /usr/bin
```

This directory contains many of the executable files with which you work in the UNIX environment. Entering **lc** with no directory name lists the contents of the current directory.

The **l** command is also useful. It is equivalent to the **ls -l** command, which produces a ''long'' listing of a directory's contents. The output of this command looks like the following listing:

```
total 338
-rw-rw-r--  1 markt pub  4448 Mar  1 09:16 1.intro.00
-rw-rw-r--  1 markt pub  4457 Mar  1 09:29 1.intro.s
-rw-rw-r--  1 markt pub 33836 Mar  1 09:30 2.concepts.00
-rw-rw-r--  1 markt pub 35096 Mar  1 12:49 2.concepts.s
-rw-rw-r--  1 markt pub 52197 Mar  1 15:09 3.basic.s
-rw-rw-rw-  1 markt pub 39835 Feb 16 11:02 4.advan.s
```

Reading from left to right, the information given for each file or directory by the **l** command includes:

- Permissions - The permissions for the first file in the above figure are **-rw-rw-r--**. This line tells you that the file is an ordinary file, that the owner and group members have read and write permission, and that all other users have read permission. For details about file and directory permissions, see ''Using File and Directory Permissions'' later in this chapter.

- Number of links - A link is a path to a file. In the above example, all of the files have one link.

- Owner - The owner's name is the login name of the person who created the file.

- Group - A group is an organization of users set up by the system administrator.

- File size in bytes

- Time of last modification

- Filename

The figure at the top lists the total number of ''blocks'' used on the disk to store these files. A single block is 512 bytes. 338 blocks, or 173056 bytes, are needed to store the files listed above.

Changing Your Working Directory

Your working directory represents your location in the filesystem. To move to a new location in the UNIX filesystem, use the **cd** command.

Entering **cd** with no arguments places you in your home directory. Try it. Enter **cd**. To be sure you are now in your home directory, enter **pwd**.

To move to a directory other than your home directory, you must specify that directory as an argument to the **cd** command. For example, enter the following command to move to */usr/bin*:

```
cd /usr/bin
```

Verify that you are in */usr/bin* by entering **pwd**.

Change to the ''root'' directory by entering the following command:

```
cd /
```

The root directory is at the ''top'' of the filesystem. All other directories are ''below'' it. Enter **lf** to examine the files and directories in the root directory. Then enter **cd** to return to your home directory. (For more information on the root directory, refer to Chapter 2 of this tutorial.)

Some shorthand notation is available to help you move quickly through the filesystem. To move up one directory from your current directory, enter:

```
cd ..
```

Enter the following command to move up two directories:

```
cd ../..
```

If you entered this latter command from your home directory, you are probably in the root directory. Verify this by entering **pwd**.

Creating Directories

To create a subdirectory in your working directory, use the **mkdir** command. Enter **cd** to move to your home directory and then enter the following command to create a subdirectory named *tempdir*:

```
mkdir tempdir
```

Verify that *tempdir* exists with the **lf** command. Change to *tempdir* with the **cd** command and verify that *tempdir* is empty with another **lf** command. Finally, use the **touch**(C) command to create two empty files in *tempdir*:

```
touch tempfile1 tempfile2
```

Enter **lf** one more time to verify that *tempfile1* and *tempfile2* were created.

You can only create subdirectories in a directory if you have write permission in that directory. If you do not have write permission and you use **mkdir** to create a subdirectory, you see the following message:

```
mkdir: cannot access directory_name
```

In this message, *directory_name* refers to the directory in which you attempted to create a subdirectory. Verify this by trying to create a subdirectory in the */etc* directory, a directory in which you probably do not have write permission:

```
mkdir /etc/temp
```

Removing Directories

Use the **rmdir** command to remove a directory. This command will not work if the directory has files or subdirectories in it. Verify this by moving to your home directory with the **cd** command and then entering the following command to remove *tempdir*, the directory created earlier in ''Creating Directories:''

```
rmdir tempdir
```

You should see the following message:

```
rmdir: tempdir not empty
```

You must remove *tempfile1* and *tempfile2* from *tempdir* before **rmdir** deletes *tempdir*. But don't remove these files just yet. They are used in another example later in this chapter.

Renaming Directories

To rename a directory, use the **mv** command. For example, **cd** to your home directory and then enter the following command to rename *tempdir*, the directory created earlier in ''Creating Directories,'' to *newdir*:

```
mv tempdir newdir
```

Verify the name change by entering **lf**. Note that the files in *newdir* are unaffected by the change. Verify this by entering the following command:

```
lf newdir
```

Copying Directories

The **copy** command copies directories. Of course, before you can copy the contents of one directory into another, you must have write permissions on the second directory.

To copy the */newdir* directory created earlier in ''Renaming Directories'' to */tmp/newdir*, enter the following command:

```
copy $HOME/newdir /tmp/newdir
```

In this command, ''$HOME'' is shorthand for the pathname of your home directory. You can use it wherever you would enter the pathname of your home directory.

When you make a copy of a directory, all or the files in the directory are copied to the new directory. To verify that the files in *$HOME/newdir* were copied to */tmp/newdir*, enter the following command:

```
lf /tmp/newdir
```

Remove */tmp/newdir* by entering the following commands:

```
rm /tmp/newdir/*
rmdir /tmp/newdir
```

The first command removes the files in */tmp/newdir*, the second command removes */tmp/newdir*. Verify that */tmp/newdir* is removed by entering the following command:

```
lf /tmp
```

Remove *$HOME/newdir* by entering the following commands:

```
rm $HOME/newdir/*
rmdir $HOME/newdir
```

Working with Files

File manipulation (creating, deleting, displaying, combining, renaming, moving, and copying) is one of the most important capabilities an operating system provides. The UNIX commands that perform these functions are described in the following sections.

Displaying File Contents

The **more** command displays the contents of a file, one screenful at a time. It cannot be used to edit files. If the file contains more than one screenful of data, you see the following prompt after each screen of text is displayed:

```
--More--(XX%)
```

XX% represents the percentage of the file displayed. Press the ⟨Return⟩ key to display another line. Press the ⟨Space⟩ (spacebar) to display another screen.

Try the following command:

```
more /etc/termcap
```

This causes the contents of */etc/termcap* to display on the screen. To quit **more** before */etc/termcap* is finished displaying, press **q** for quit.

The **more** command does not allow you to scroll backward, toward the beginning of the file. However, you can search forward for patterns with **more** by using the slash (/) command. For example, enter the following commands to search for a line containing ''process'' in */etc/termcap*:

```
more /etc/termcap
/process
```

You see the following message at the top of the screen:

```
...skipping
```

If the pattern is found, it is displayed two lines below this message. If the pattern is not found, ''Pattern not found'' is displayed.

If you are looking at a file with **more** and decide that you want to edit the file, you can invoke the **vi** editor by pressing **v**. Of course, you must have write permission on a file before you can edit it with **vi** or any other text editor. To display the file's contents, you only need read permission.

You will often use **more** in pipes. For example, **more** is useful when you want to list the contents of a directory in long format. Enter the following command to display a long listing of the contents of */bin*, one screenful at a time:

```
l /bin | more
```

(For more information on pipes, refer to Chapter 2 of this tutorial.)

The **head** and **tail** commands display the beginning and the end of a file, respectively. With no options, they display the first or last 10 lines. Enter the following command to display the last 10 lines of */etc/termcap*:

```
tail /etc/termcap
```

You can specify exactly how many lines you want displayed. Enter the following command to display the first 20 lines of */etc/termcap*:

```
head -20 /etc/termcap
```

Enter the following command to display the last 20 lines of */etc/termcap*:

```
tail -20 /etc/termcap
```

The **cat** command also displays the contents of a file. Unlike **more**, **cat** continuously scrolls the file until you stop the scroll with ⟨CTL⟩s. ⟨CTL⟩q continues scrolling. Scrolling stops automatically when the end of the file is reached. To stop scrolling before the end of the file, press **INTERRUPT**, which is the ⟨DEL⟩ key on most keyboards.

Enter the following command to display the contents of */etc/termcap*. Use ⟨CTL⟩s and ⟨CTL⟩q to stop and start the scrolling and **INTERRUPT** to halt the scrolling before the end of the file is reached:

```
cat /etc/termcap
```

Listing Invisible (Dot) Files

Filenames beginning with a period (dot) are invisible; the normal listing commands like **l**, **lc**, etc., do not display these files. These commands include a **-a** option that lists **all** files. For example, the command:

lc -a

might display a list of files like this:

```
 .       ..      .cshrc  .login  prints   quotes  globals
```

The "." and ".." refer to the present and upper directories, respectively. The files *.cshrc* and *.login* are invisible files.

Deleting Files

The **rm** command is used to delete files. We have used it throughout this chapter to delete various files. Use **cd** to change to your home directory and enter the following command to create three new files:

touch tempfile1 tempfile2 tempfile3

Delete *tempfile3* by entering the following command:

rm tempfile3

The **-i** option allows you to remove files interactively by asking you if you really want to delete each of the files specified on the command line. If you press **y** followed by a ⟨Return⟩, the given file is removed. If you press **n,** the file is left untouched. This option is useful when removing files from a directory that contains many files. It helps you avoid erasing files accidentally that you really want to keep.

Experiment with the **-i** option by entering the following command:

rm -i tempfile1 tempfile2

Note that you can place several filenames on the **rm** command line. This is true for most UNIX commands. You can also use wildcard characters. For example, instead of entering the above command, you could enter the following:

```
rm -i tempfile*
```

(The use of wildcard characters on the command line is discussed in Chapter 2 of this tutorial.)

Combining Files

In addition to displaying files, the **cat** command can be used to combine several existing files into a single new file. This is done by redirecting the output of **cat** into a new file. The greater-than sign (>) is used for redirection. If the new file does not exits, it is created automatically. (If you are not familiar with redirection, see Chapter 2 of this tutorial.)

Use **cd** to move to your home directory and enter the following command to combine */etc/motd* and */etc/default/tar* into a file named *catfile*:

```
cat /etc/motd /etc/default/tar > catfile
```

Now display the contents of the new file *catfile* with the **more** command:

```
more catfile
```

The symbol >> can be used with **cat** to *append* one file to the end of another file. For example, to append the contents of */etc/motd* to *catfile*, enter the following command:

```
cat /etc/motd >> catfile
```

The contents of */etc/motd* should now be placed at the beginning and at the end of *catfile*. Verify this with the following *head* and *tail* commands:

```
head -20 catfile
tail -20 catfile
```

Renaming Files

The **mv** command is used to move files around the UNIX filesystem and also to rename files. Use **cd** to move to your home directory. Rename *catfile*, created earlier in ''Combining Files,''to *catfile2* by entering the following command:

```
mv catfile catfile2
```

After this move is completed, *catfile* no longer exists. The file *catfile2* exists in its place. Verify this by entering the following command:

```
lc
```

Moving Files

To move a file into another directory, give the name of the destination directory as the final name in the **mv** command. You do not need to specify the destination filename. For example, enter the following command to move *catfile2,* created earlier in ''Renaming Files,'' to the */tmp* directory:

```
mv $HOME/catfile2 /tmp
```

To be sure that *catfile2* is in */tmp* and not in the current directory, enter the following command:

```
lc . /tmp
```

(Remember that you can enter more than one argument on most command lines, and that the dot (**.**) stands for the current directory.)

Finally, move *catfile2* back to the current directory by entering the following command:

```
mv /tmp/catfile2 .
```

The **mv** command always checks to see if the last argument is the name of a directory. If it is, all files designated by filename arguments are moved into that directory. However, if you do not have write permission on the directory to which you are attempting to move files, the move fails.

Copying Files

The **cp** command is used to copy files. There are two forms of the **cp** command, one in which files are copied into a directory and another in which a file is copied to another file.

Use **cd** to change to your home directory. Then enter the following command to copy the contents of *catfile2*, created earlier in ''Renaming Files,'' to *catfile3*:

```
cp catfile2 catfile3
```

You now have two files with identical contents. To copy *catfile2* and *catfile3* to the */tmp* directory, enter the following command:

```
cp catfile2 catfile3 /tmp
```

This last command can be simplified by using a wildcard character:

```
cp catfile* /tmp
```

Like the **mv** command, **cp** always checks to see if the last argument is the name of a directory, and, if so, all files designated by filename arguments are copied into that directory. However, unlike the **mv** command, **cp** leaves the original file untouched. There should now be two copies of *catfile2* and *catfile3* on the system, one copy of each in the current directory and one copy of each in */tmp*.

Finding Files

A UNIX filesystem can contain thousands of files. Because of this, files can often get lost. The **find** command is used to search the filesystem for files. The command has the form:

find *pathname* **-name** *filename* **-print**

The *pathname* is the pathname of the directory that you want to search. The search is recursive; it starts at the directory named and searches downward through all files and subdirectories under the named directory.

The **-name** option indicates that you are searching for files that have a specific *filename*. The **-print** option indicates that you want to print the pathnames of all the files that match *filename* on your screen.

Enter the following command to search all directories and subdirectories for *catfile2*, the file created earlier in ''Renaming Files:''

```
find / -name catfile2 -print
```

It may take a few minutes for this command to finish executing. The output of this **find** command should indicate that there are at least two occurrences of *catfile2*, one in */tmp* and one in your home directory. Remove *catfile2* and *catfile3* from */tmp* and from your home directory by entering the following command:

```
rm /tmp/catfile* $HOME/catfile*
```

Editing Files with vi

The **vi** text editor is a full-screen editor included with UNIX operating systems. The sections that follow briefly explain how to use **vi**. For a more complete discussion, see the *User's Guide*.

Entering Text

Change to your home directory with the **cd** command and enter the following command to create a file called *tempfile*:

```
vi tempfile
```

A message appears indicating that you are creating a new file. You are then placed in **vi**.

There are two modes in **vi**: *Insert mode* and *Command mode*. Use Insert mode to add text to a file. Use Command mode to edit existing text in a file. Since *tempfile* is empty, press **i** to enter Insert mode.

Enter the following lines of text, pressing ⟨Return⟩ after each line. If you make a mistake typing, use the ⟨BKSP⟩ key to erase the mistake and continue typing:

```
This tutorial is very, very helpful.
It makes learning to use a UNIX system easy.
I'm glad I have this tutorial.
```

After you enter the last line, press the ⟨ESC⟩ key. It takes you out of Insert mode and places you in Command mode.

Moving the Cursor

Although many cursor-movement commands are available in **vi**, only the four basic ones are discussed here:

h	When you are in Command mode, pressing the **h** key moves the cursor one character to the left.
l	When you are in Command mode, pressing the **l** key moves the cursor one character to the right.

k When you are in Command mode, pressing the **k** key moves the cursor up one line.

j When you are in Command mode, pressing the **j** key moves the cursor down one line.

Experiment with these cursor-movement keys on the text you entered. Note that, if your keyboard has arrow keys, these usually perform in the manner of **h, l, k** and **j**.

Deleting Text

Deleting text with **vi** is very easy. Different commands allow you to delete characters, words and entire lines.

To delete a single character, place the cursor on that character with the cursor-movement keys and press the **x** key. Experiment with the **x** key by deleting the comma in the first line.

To delete a word, place the cursor on the first character of the word and press **dw** (press **d**, release it, and press **w**). Experiment with this by placing the cursor on the first character of ''very'' in the first line and pressing **dw**.

To delete an entire line, place the cursor anywhere on that line and press **dd** (press **d**, release it, and press **d** again). Experiment with this by placing the cursor on the third line and pressing **dd**. Your file should now contain the following text:

```
This tutorial is very helpful.
It makes learning to use the system easy.
```

Inserting Text

The **i** and **o** keys can be used to insert text. We have already used the **i** key to enter text in an empty file. To enter additional text on an existing line, move the cursor to the point where you want the new text to begin, press **i** to enter Insert mode, enter the text, and press ⟨ESC⟩ to return to Command mode. For example, move the cursor to the "e" in "easy" in the second line, press **i**, enter the word **very**, press the ⟨Space⟩, and press ⟨ESC⟩ to return to Command mode. The second line should now be:

```
It makes learning to use the system very easy.
```

The **o** key can be used to insert a new line. To use it, move the cursor to the line *directly above* the place in the file where the new line is to be inserted and press **o**. A new line is inserted, with the cursor placed at the beginning. You are also automatically placed in Insert mode. Try this by moving the cursor to the second line of *tempfile* and press **o**. Now enter more text. Press ⟨ESC⟩ when you are finished.

Leaving vi

Most of the time, you will want to save your file before leaving **vi**. To do this, enter Command mode and type **:x**. This command saves the file you are editing and returns you to the UNIX prompt.

In some cases, you will want to leave **vi** without saving your work. To do this, enter Command mode and type **:q!**. This command returns you to the UNIX prompt, without saving the changes that you made to your file.

Leave *tempfile* by pressing **:x**. Re-enter *tempfile* by entering the following command:

```
vi tempfile
```

Insert some text using either the **i** or the **o** key, press ⟨ESC⟩ and then enter **q!** to quit without saving your changes. Display *tempfile* by entering the following command:

```
cat  tempfile
```

You will notice that the last set of changes you made do not appear. Remove *tempfile* by entering the following command:

```
rm  tempfile
```

Printing Files

To print files, use the **lp** command. This is one of a group of commands known as the ''lineprinter'' commands. The lineprinter commands are easy to use and very flexible. With a few simple commands, you can print multiple copies of a file, cancel a print request, or ask for a special option on a particular printer. Check with your system administrator to find out what lineprinters and printer options are available on your system.

Using lp

Use **cd** to change to your home directory and enter the following command to create a file with which you can experiment:

```
cp /etc/motd $HOME/printfile
```

This command places a copy of */etc/motd* in your home directory, naming it *printfile*. The file */etc/motd* is the ''message of the day file.'' Its contents are displayed every time you log in to a UNIX system.

A directory must be ''publicly executable'' before you can use **lp** to print any of the files in that directory. This means that other users must have execute permissions on the directory. Enter the following command to make your home directory publicly executable:

```
chmod o+x $HOME
```

(See ''Using File and Directory Permissions'' later in this chapter for more information on **chmod**(C).)

Enter the following command to print *printfile*:

```
lp printfile
```

This command causes one copy of *printfile* to print on the default printer on your system. A banner page might be printed along with the file. Note that you can print several files at once by putting more than one name on the **lp** command line.

When you print with **lp**, a "request ID" is displayed on your screen. A request ID might look like the following:

```
    pr4-532
```

The first part (pr4) is the name of the printer on which your file is printed. The second part (532) identifies your job number. Should you later wish to cancel your print request or check its status, you will need to remember your request ID. (Cancelling and checking on print requests are discussed below.)

You can also use **lp** with pipes. For example, enter the following command to sort and then print a copy of */etc/passwd*, the file that contains system account information:

```
    sort /etc/passwd | lp
```

(For more information on **sort**(C), see "Sorting Files" later in this chapter.)

Using lp Options

The **lp** command has several options that help you control the printed output. You can specify the number of copies you want printed by using the number option, **-n**. For example, to print two copies of *printfile*, enter:

```
    lp printfile -n2
```

Several different printers are often attached to a single UNIX system. With the **-d** option, you can specify the printer on which your file is to be printed. To print two copies of *printfile* on a printer named *laser*, enter:

```
    lp printfile -n2 -dlaser
```

Check with your system administrator for the names of the printers available on your system.

Cancelling a Print Request

Use the **cancel** command to cancel any of your print requests. With this command, you can only cancel your own requests. The system administrator is the only person allowed to cancel the requests of other users. If the system administrator cancels one of your print requests, you are automatically notified via mail.

The **cancel** command takes as its argument the request ID. For example, to stop printing one of your files with a request ID of *laser-245*, you would enter:

```
cancel  laser-245
```

Experiment with **cancel** by using **lp** to print *printfile* and then using **cancel** to cancel the print request. When you are finished, enter the following command to remove *printfile*:

```
rm  printfile
```

You can also use the **cancel** command to stop whatever is currently printing on a particular printer. For example, to cancel whatever file is currently printing on the printer *laser*, you would enter the following command:

```
cancel  laser
```

If you cancel a file that does not belong to you, mail reporting that the print request was canceled is automatically sent to the file's owner.

Finding Out the Status of a Print Request

Use the **lpstat** command to check on the status of your print request. To use it, simply enter the following:

```
lpstat
```

The **lpstat** command produces output like the following:

```
prt1-121    cindym    450   Dec 15 09:30
laser-450   cindym   4968   Dec 15 09:46
```

Note that entering **lpstat** with no options displays information on your files only, not those of other users. To generate a report for all users on your computer, use **lpstat** with the **-o** option. Nothing is displayed by the **lpstat** command if the print job is complete.

Note

> Depending on how your system administrator has set up your system, you may not be allowed to see other print jobs. Refer to ''Using a Trusted System'' in the *User's Guide* for details.

The first column of the **lpstat** output shows the request ID for each of your files being printed. The second column is your login name. In the third column, the number of characters to be printed is shown, and the fourth column lists the dates and times the print requests were made.

To learn the status of a particular file, use the **lpstat** command with the file's request ID. For example, to find out the status of a file with the request ID of *laser-256*, you would enter the following command:

```
lpstat laser-256
```

The status of that file only is displayed.

You can also request the status of various printers on your system by using the **-p** option or by giving the name of the particular printer you are interested in. Enter the following command to find out the status of all the printers on your system:

```
lpstat -p
```

To find out the status of a printer named *laser*, you would enter the following:

```
lpstat -plaser
```

The request ID and status information for each file currently waiting to be printed on *laser* is displayed.

Processing Text Files

UNIX systems include a set of utilities that let you process information in text files. These utilities enable you to compare the contents of two files, sort files, search for patterns in files, and count the characters, words, and lines in files. These utilities are described below.

Comparing Files

The **diff** command allows you to compare the contents of two files and to print out those lines that differ between the files. To experiment with **diff**, use **vi** to create two files to compare. The files will be *men* and *women*. First **cd** to your home directory. Then enter the following command at the UNIX prompt:

```
vi men
```

When you are placed in **vi**, press the **i** key to enter Insert mode, and then type the following lines:

```
Now is the time for all good men to
Come to the aid of their party.
```

Press ⟨ESC⟩ to return to Command mode and save *men* by entering **:w**. While still in Command mode, enter the following command to create *women*:

```
:n women
```

You see the following message:

```
"women" No such file or directory
```

You are then placed in *women*. Press **i** to enter Insert mode and then enter the following lines:

```
Now is the time for all good women to
Come to the aid of their party.
```

Press ⟨ESC⟩ to return to Command mode, then **:x** to save *women* and leave **vi**. You have now created *men* and *women*.

Enter the following command to compare the contents of these two files:

diff men women

This **diff** command should produce the following output:

```
1c1
< Now is the time for all good men to
---
> Now is the time for all good women to
```

The lines displayed are the lines that differ from one another in the two files.

Sorting Files

One of the most useful file processing commands is **sort**. When used without options, **sort** alphabetizes lines in a file, starting with the leftmost character of each line. These sorted lines are then output to the screen, or to a file if redirection is used on the **sort** command line. This command does not affect the contents of the actual file.

Enter the following command to display an alphabetized list of all users who have system accounts:

sort /etc/passwd

The **sort** command is useful in pipes. Enter the following command to display an alphabetized list of users who are currently using the system:

who | sort

Searching for Patterns in a File

The **grep** command selects and extracts lines from a file, printing only those lines that match a given pattern. Enter the following command to print out the lines in */etc/passwd* that contain your login information. There will probably be only one such line:

grep *login* /etc/passwd

Be sure to replace *login* in this command with your login name. Your output should be similar to the following:

```
markt:+:6005:104:Mark Taub, Docland:/u/markt:/bin/csh
```

Note that whenever wildcard characters are used to specify a **grep** search pattern, the pattern should be enclosed in single quotation marks ('). Note also that the search pattern is case sensitive. Searching for ''joe'' will not yield lines containing ''Joe''.

As another example, assume that you have a file named *phonelist* that contains a name followed by a phone number on each line. Assume also that there are several thousand lines in this list. You can use **grep** to find the phone number of someone named Joe, whose phone number prefix is 822, by entering the following command:

```
grep ´Joe´ phonelist | grep ´822-´ > joes.number
```

The **grep** utility first finds all occurrences of lines containing the word ''Joe'' in the file *phonelist*. The output from this command is then filtered through another **grep** command, which searches for an ''822-'' prefix, thus removing any unwanted ''Joes.'' Finally, assuming that a unique phone number for Joe exists with the ''822-'' prefix, that name and number are placed in the file *joes.number*.

Two other pattern searching utilities are available. These are **egrep** and **fgrep.** Refer to **grep**(C) in the *User's Reference* for more information on these utilities.

Counting Words, Lines, and Characters

The **wc** utility is used to count words in a file. Words are presumed to be separated by punctuation, spaces, tabs, or newlines. In addition to counting words, **wc** counts characters and lines.

Use **cd** to change to your home directory. Then enter the following command to count the lines, words, and characters in the file *men,* created earlier in ''Comparing Files:''

```
wc men
```

The output from this command should be the following:

```
  2     16     68 men
```

The first number is the number of lines in *men*, the second number is the number of words and the third number is the number of characters. Remove *men* and *women* by entering the following command:

```
rm *men
```

To specify a count of characters, words, or lines only, you must use the **-c**, **-w**, or **-l** option, respectively. For example, enter the following command to count the number of users currently logged onto the system:

```
who | wc -l
```

The **who** command reports on who is using the system, one user per line. The **wc -l** command counts the number of lines reported by the **who** command. This is the number of users currently on the system.

Using File and Directory Permissions

UNIX systems allow the owner of a file or directory to restrict access to that file or directory. This is done with permission settings. Permissions on a file limit who can read, write and/or execute the files. Permissions on a directory limit who can **cd** to the directory, list the contents of the directory, and create and delete files in the directory.

To determine the permissions associated with a given file or directory, use the **l** command. Use **cd** to change to your home directory and then enter **l** to get a long listing of the files in this directory.

Permissions are indicated by the first 10 characters of the output of the **l** command. The first character indicates the type of file and must be one of the following:

-	Indicates an ordinary file.
b	Indicates a block special device such as a hard or floppy disk. Hard and floppy disks can be treated as both block and character special devices.
c	Indicates a character special device such as a lineprinter or terminal.
d	Indicates a directory.
m	Indicates a shared data file.
n	Indicates a name special file.
p	Indicates a named pipe.
s	Indicates a semaphore.

From left to right, the next nine characters are interpreted as three sets of three permissions. Each set of three indicates the following permissions:

- Owner permissions,
- Group permissions, and
- All other user permissions.

Within each set, the three characters indicate permission to read, to write, and to execute the file as a command, respectively. For a directory, ''execute'' permission means permission to search the directory for any files or directories.

Ordinary file permissions have the following meanings:

r	The file is readable.
w	The file is writable.
x	The file is executable.
-	The permission is not granted.

For directories, permissions have the following meanings:

r	Files can be listed in the directory; the directory must also have ''x'' permission.
w	Files can be created or deleted in the directory. As with ''r'', the directory itself must also have ''x'' permission.
x	The directory can be searched. A directory must have ''x'' permission before you can move to it with the **cd** command, access a file within it, or list the files in it. Remember that a user must have ''x'' permission to do anything useful to the directory.

The following are some typical directory permission combinations:

d--------	No access at all. This is the mode that denies access to the directory to all users but the super user.
drwx------	Limits access to the owner. The owner can list the contents of this directory and the files in it (if they have appropriate permissions), **cd** to the directory, and add files to, and delete files from, the directory. This is the typical permission for the owner of a directory.
drwxr-x---	In addition to allowing the owner all of the above access permissions, this setting allows group members to list the contents of this directory and files within it and to **cd** to this directory. However, group members cannot create files in, or delete files from, this directory. This is the typical permission an owner gives to others who need access to files in his or her directory.
drwxr-x--x	In addition to allowing the owner and the group all of the above access permissions, this setting allows users other than the owner or members of the group to **cd** to this directory. However, because the **r** is not set for others, other users cannot list the contents of this directory with any of the **ls** commands. This mode is rarely used, but it can be useful if you want to give someone access to a specific file in a directory without revealing the presence of other files in the directory.

The */etc* directory contains files whose permissions vary. Examine the permissions of the files in this directory by entering the following command:

```
l /etc | more
```

Changing File Permissions

The **chmod** command changes the read, write, execute, and search permissions of a file or directory. It has the form:

chmod *instruction filename*

The *instruction* argument indicates which permissions you want to change for which class of users. There are three classes of users, and three levels of permissions. The users are specified as follows:

u	User, the owner of the file or directory.
g	Group, the group the owner of the file belongs to.
o	Other, all users of the system who are not in u or g.
a	All users of the system.

The permissions are specified as follows:

r	Read, which allows permitted users to look at but not change or delete the file.
w	Write, which allows permitted users to change or even delete the file.
x	Execute, which allows permitted users to execute the file as a command.

Use **cd** to move to your home directory. Then enter the following command to create *tempfile*:

```
touch tempfile
```

The permissions on *tempfile* are probably:

```
-rw-r--r--
```

Verify this by entering the following command:

```
l tempfile
```

Enter the following command to give yourself (the file's owner) execute permissions on *tempfile*:

```
chmod  u+x  tempfile
```

Verify the permissions change with the **l** command. (Of course, since *tempfile* is neither a binary nor a script, having execute permission on it is meaningless.)

Enter the following command to give the group and other users write permission on *tempfile*:

```
chmod  go+w  tempfile
```

Verify the permissions change with the **l** command.

The **chmod** command also allows you to remove permissions. For example, enter the following command to prohibit others from writing to *tempfile*:

```
chmod  o-w  tempfile
```

Remove *tempfile* with the following command:

```
rm  tempfile
```

Changing Directory Permissions

Directories also have an execute permission, even though they cannot be executed in the same way that a script or binary file can. For directories, the execute attribute is needed in order to do any useful work in a directory. Users who do not have execute permission for a directory cannot **cd** to the directory, list the names of files in the directory, or copy files to or from the directory.

The permissions on your home directory are probably set to the following:

drwxr-xr-x

Verify this by entering the following command:

```
l -d $HOME
```

You probably see output like the following:

```
drwxr-xr-x   4 markt    pub         240 Feb 10 09:09 /u/markt
```

This setting allows you, the directory's owner, to **cd** to the directory, to list the contents of the directory and of the files within it (if the file permissions also allow), and to create and delete files in the directory. This setting also allows members of the group and other users to **cd** to the directory, to list the directory's contents and also the contents of files within the directory, if file permissions allow.

To deny any useful access to others, enter the following command:

```
chmod o-x $HOME
```

Verify that the permissions were changed with the following command:

```
l -d $HOME
```

Your output should look like the following:

```
drwxr-xr--    4 markt     pub         240 Feb 10 09:09 /u/markt
```

Now, only you and members of the group have access to your directory. If you want to restore access to your home directory to other users, enter the following command:

```
chmod o+x $HOME
```

Summary

Directories and *subdirectories* are created to organize the UNIX filesystem. Each directory or subdirectory can contain both files and other directories, and can be accessed by anyone with read permission for the file or directory in question.

Whenever you move to a different directory, it becomes, by definition, your *working directory*. There are UNIX commands for:

- Displaying the name of the working directory,
- Creating directories,
- Removing directories,
- Renaming directories,
- Copying directories,
- Listing directory contents,
- Changing your working directory.

The *files* that reside in directories are the most basic means of storing data on a UNIX system. Each file ''belongs'' to a directory somewhere on the system; it is impossible for a file to exist without a ''parent'' directory.

You can manipulate files in the same ways that you can manipulate directories (described above). There are UNIX commands for:

- Creating files,
- Deleting files,
- Displaying files,
- Combining files,
- Renaming files,
- Moving files,
- Copying files,
- Printing files.

You can also set the access permissions for any of your files or directories. This feature gives you control over who can read, edit, and execute your files and directories.

Summary

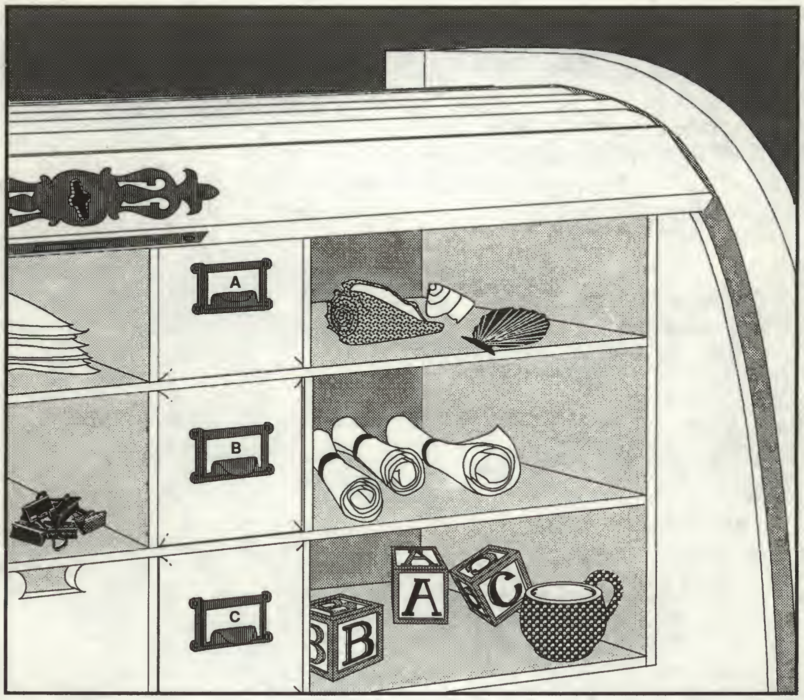
UNIX systems provide a full-screen text editor that is very useful for editing files. This editor, called **vi**, can be called up from anywhere on the system. While you are in **vi**, you can add or delete text to or from a file, change the existing text, or create a new file.

If you have access to a printer, you can produce a hard copy of any of your files. If you have more than one printer, you can choose which one to use. You can also specify the number of copies to be printed, check the status of a print request, and cancel a print request at any time.

UNIX systems provide a set of *utilities* that let you process the information in text files. These utilities enable you to:

- Count the characters in a file,
- Count the words in a file,
- Count the lines in a file,
- Compare the contents of two files,
- Sort files
- Search for patterns in a file.

For a complete description of the commands and utilities presented in this chapter, see the *User's Guide* and the *User's Reference*.



Chapter 5

Housekeeping

Introduction

This chapter explains how to perform "housekeeping" tasks on a UNIX system. Housekeeping tasks are maintenance tasks that you perform periodically, tasks that provide you with information about system resources, as well as tasks that let you operate more efficiently in the UNIX environment. This chapter explains how to perform the following housekeeping tasks:

- Create backups of valuable files and directories,
- Extract files from backup media,
- Make copies of floppy diskettes,
- Find out who is on the system,
- Determine how much disk space is used/free,
- Run a command in the background,
- Delay and kill the execution of commands,
- Use the shell programming language to automate tedious tasks.

This chapter is designed as a tutorial. The best way to use this chapter is to read it at your terminal, entering commands as instructed in the examples.

None of the commands described in this chapter is described in great detail. For a complete explanation of each command, refer to the *User's Reference*.

Making Backups

Backing up all the filesystems on a UNIX system is often the responsibility of the system administrator. However, individual users often find it useful to create periodic backups of the particular files with which they work. These backups are created with the **tar** command.

Floppy diskettes are the backup media used most often. Cartridge tapes are also used for backups. However, floppy diskettes and some cartridge tapes must be formatted before they can be used. The sections that follow explain how to format floppy diskettes and certain cartridge tapes and how to use **tar** to create backups.

Formatting Diskettes and Tapes

To format a 5.25 inch 1.2 megabyte diskette (double-sided, double-density) in the first floppy drive, enter the following command:

```
format
```

You are instructed to insert a diskette into the drive and press ⟨Return⟩.

If your first floppy drive is a 5.25 inch double-sided high-density drive, enter the following command to format a 1.2 megabyte diskette in it:

```
format /dev/rfd096ds15
```

To format a 3.5 inch 720K micro-floppy diskette in the first floppy drive, enter:

```
format /dev/rfd096ds9
```

By replacing the **0** that follows the **rfd** in these commands with a **1**, you can format diskettes in a second floppy drive on your system.

It is not necessary to format most cartridge tapes. However, cartridge tapes used with the *mini tape drives* must be formatted. To format one of these cartridges, enter:

```
format /dev/rctmini
```

Using tar to Create Backups

The **tar** command is used to create backups. The syntax of **tar** is:

tar [*key*] [*files*]

The *key* argument controls the actions of **tar**. The *files* argument specifies which files to back up.

The *keys* used most often are:

c	Creates a backup.
x	Extracts files from backup media.
t	Lists the contents of backup media.
v	Displays the name of each file being processed.
f	Creates backups on a specified device.
u	Creates a backup only if the file has not been backed up before, or if the file has been modified since the last backup.

Creating a Backup

The steps outlined below explain how to back up all of the files in your home directory onto floppy diskettes. Experiment with **tar** by following these steps.

To back up a different directory, merely **cd** to that directory and follow these same steps. To back up onto a tape, substitute the special device file associated with the tape, such as */dev/rctmini* or */dev/rct0*, for the floppy device file listed in these commands.

1. Log in on the console. This allows you to work within arm's reach of the floppy drive.

2. Determine how many floppy diskettes you need and format that many, using the **format** command as described in ''Formatting Diskettes and Tapes'' above. To determine how many diskettes to format, enter the following command:

   du -a

   Your output should look like the following:

```
12     ./1.intro.s
74     ./2.concepts.s
14     ./2.concepts.err
0      ./.err
60     ./5.house.s
32     ./3.log.s
2      ./err
2      ./0.title
30     ./6.desk.s
112    ./4.files.s
12     ./4.files.err
4      ./3.log.err
356    .
```

   The number at the bottom represents the total number of 512 byte blocks used by the files in the current directory. In this example, you need a total of 512 x 356 bytes, or roughly 183 kilobytes. You only need to format a single floppy disk to backup this directory.

3. Enter the following command to back up all of the files in your home directory to 5.25 inch 1.2 megabyte floppy diskettes in the first floppy drive:

   tar cvf /dev/fd096ds15 .

   If **tar** requires more than 1 diskette, you are prompted to insert another ''volume.'' Insert another diskette if you see this prompt. The **tar** command is finished when the shell prompt reappears.

To make a backup of just a single file onto a 1.2 megabyte diskette, enter:

```
tar cvf /dev/fd096ds15 ./filename
```

Note that *filename* is preceded by a dot and a slash (./). This tells **tar** to treat *filename* as a "relative" rather than an "absolute" filename. (For more information, see **tar**(C).)

tar will recreate, on the backup media, all subdirectories of the directory you are backing up. Thus, if you have a */bin* directory in your home directory, **tar** will create a backup of it, and all the files in it, on the backup media.

Listing the Contents of Backups

To list the contents of a 5.25 inch 360 kilobyte tar-floppy in the first floppy drive, enter:

```
tar tvf /dev/fd048ds9
```

To list the contents of a 5.25 inch 1.2 megabyte tar-floppy in the first floppy drive, enter:

```
tar tvf /dev/fd096ds15
```

To list the contents of a 3.5 inch tar micro-floppy in the first floppy drive, enter:

```
tar tvf /dev/fd0135ds9
```

Experiment with this **tar** option by placing the backup of your home directory that you created in the previous section into the first floppy drive. Enter the appropriate command to list its contents.

Extracting Files from Backups

We recommend that you extract files from backup media into a temporary directory on the hard disk. Once in the temporary directory, you can use the **mv** command to move the extracted files to the correct location on the filesystem. The reason for using a temporary directory is that **tar** will overwrite any files on the hard disk that have the same name as the file being extracted from the backup media. This can result in files being overwritten accidentally.

To extract all of the tar-format files from a 5.25 inch 360 kilobyte diskette in the first floppy drive, enter:

```
tar xvf /dev/fd048ds9
```

To extract tar-format files from a 5.25 inch 1.2 megabyte floppy diskette in the first floppy drive, enter:

```
tar xvf /dev/fd096ds15
```

To extract a single file from a 1.2 megabyte floppy in the first floppy dirve, enter:

tar xvf /dev/fd096ds15 *./filename*

Note that *filename* is preceded by a dot followed by a slash (./). This assumes that *filename* was copied to the **tar** floppy diskette with a dot (.), as in the examples in ''Using tar to Create Backups.'' When you copy files to a **tar** floppy with a dot, the ./ is prefixed to the filenames. Because you must enter a filename exactly as it appears on the floppy when extracting a file with **tar**, you must enter **./filename** if *filename* was copied to the **tar** floppy with a dot.

Experiment with these **tar** commands by placing the diskette containing the backup of your home directory (that you created in ''Using tar to Create Backups'') into the first floppy drive. Follow these steps:

1. Enter the following command to change to */tmp*:

   cd /tmp

2. Create a subdirectory in */tmp* by entering:

   mkdir *login*

   Replace *login* with your login name.

3. Enter:

   cd *login*

4. If you are a Bourne shell user, and if your first floppy drive is a 1.2 megabyte drive, experiment with extracting a single file by entering the following command to extract *.profile*:

   ```
   tar xvf /dev/fd096ds15 ./.profile
   ```

   If you are a C-shell user, enter:

   ```
   tar xvf /dev/fd096ds15 ./.login
   ```

   If your floppy drive is not a 1.2 megabyte drive, enter the appropriate special device filename.

5. To check that the files were actually copied to the hard disk, enter:

   ```
   lc -a
   ```

   The **-a** option tells **lc** to show hidden files, those whose filenames begin with a dot (.).

6. To experiment with extracting all of the files on a **tar** floppy, enter the following command if your first floppy drive is a 1.2 megabyte drive:

   ```
   tar xvf /dev/fd096ds15
   ```

   If your floppy drive is not a 1.2 megabyte drive, enter the appropriate special device filename.

Shorthand tar Notation

The **tar** command also provides shorthand notation. This notation allows you to specify numbers in place of full special device filenames. The file */etc/default/tar* assigns numbers to the various floppy and tape devices. Enter the following to display the contents of */etc/default/tar*:

```
more /etc/default/tar
```

Your output should look similar to the following:

```
#   device                    block   size   tape
archive0=/dev/rfd048ds9       18      360    n
archive1=/dev/rfd148ds9       18      360    n
archive2=/dev/rfd096ds15      10      1200   n
archive3=/dev/rfd196ds15      10      1200   n
archive4=/dev/rfd096ds9       18      720    n
archive5=/dev/rfd196ds9       18      720    n
archive6=/dev/rfd0135ds18     18      1440   n
archive7=/dev/rfd1135ds18     18      1440   n
archive8=/dev/rct0            20      0      y
archive9=/dev/rctmini         20      0      y
#   The default device...
archive=/dev/rfd096ds15       10      1200   n
```

This file assigns **0** to the first 360 kilobyte drive, **1** to the second 360 kilobyte drive, **2** to the first 1.2 megabyte drive, **3** to the second 1.2 megabyte drive, and so forth.

To copy all the files in the current directory to 5.25 inch 360 kilobyte diskettes in the first floppy drive, enter:

tar cv .

(The default device is device **0**. You do not have to specify the default device name in the command in order to use the default device.)

To copy all the files in the current directory to 5.25 inch 1.2 megabyte diskettes in the first floppy drive, enter:

tar cv2 .

To extract *filename* from a 3.5 inch 720 kilobyte tar micro-floppy in the first floppy drive, enter:

```
tar xv4 ./filename
```

Note that the version of */etc/default/tar* on your system may differ from that shown here. This is because your system administrator may have edited it. Before you use this shorthand notation, double-check the assignments in the */etc/default/tar* file on your system.

Copying Diskettes

To protect against the loss of data stored on floppy disks, you can use the **diskcp**(C) command to make copies of your floppy diskettes.

You must copy information onto formatted disks. If you format floppies on a UNIX system, you can use them over again without reformatting. If you have disks that have been formatted under another operating system, you must reformat them on a UNIX system before you can use them to make copies of UNIX disks. Be aware that floppies formatted under some operating systems cannot be used under other operating systems, even with reformatting.

The **diskcp** command can format floppies before making copies. The following steps explain how to use **diskcp:**

1. Place the disk that you want to copy, the ''source'' floppy, in your primary floppy drive. If you created a backup of your home directory, as instructed in ''Using tar to Create Backups,'' experiment with **diskcp** by using this backup. Place it in the floppy drive.

2. Place another floppy in the other drive. This floppy is the ''target'' disk. Note that any information already on the target disk will be destroyed.

   If you have only one disk drive, leave the source floppy in the drive. It will be copied to the computer's hard disk, and from there to the target floppy. You will be prompted to remove the source floppy and insert the target floppy.

3. To format the target floppy as a 1.2 megabyte floppy before copying, enter the command:

   ```
   diskcp -f
   ```

   If you do not need to format the target floppy, and if the floppy you are copying is a 1.2 megabyte floppy, enter:

   ```
   diskcp
   ```

If your computer has dual floppy drives, enter the following command to copy a 1.2 megabyte source floppy directly on a formatted target floppy:

```
diskcp -d
```

4. Follow the instructions as they appear on your screen. Note that, with a single drive system, you are prompted to remove the source disk and insert the target disk.

5. If you made a copy of the backup of your home directory, place this floppy in the first floppy drive and verify that the files were copied correctly by entering:

   ```
   tar tvf /dev/fd096ds15
   ```

   If your floppy is a 360 kilobyte floppy, enter:

   ```
   tar tvf /dev/fd048ds9
   ```

   Note that you can use the shorthand **tar** notation in these commands, as explained in ''Shorthand tar Notation'' above.

Getting Status Information

Because a UNIX system is a large, self-contained computing environment, there are many things that you may want to find out about the system itself, such as who is logged in and how much disk space is available. The sections that follow explain how to do this.

Finding Out Who Is on the System

The **who** command lists the names, terminal line numbers, and login times of all users currently logged onto the system. Enter the following command to find out who is on your system:

who

Your output should look like the following:

```
arnold  tty1a  Apr7  10:02
daphne  tty1b  Apr7  07:47
elliot  tty1c  Apr7  14:21
ellen   tty2a  Apr7  08:36
gus     tty2b  Apr7  09:55
adrian  tty2c  Apr7  14:21
```

The **finger** command can also be used to find out who is on the system. It provides more detailed information. To use it, simply enter **finger**.

Determining Disk Usage

The **df** command displays figures on disk free space. When used without options, it reports the number of free blocks and inodes in all the filesystems on your computer. A block is 512 bytes, and an inode is a data structure reserved for information about a file. Enter the following command to display disk free space figures:

df

Your output should look like the following:

```
/       (/dev/root ):     5956 blocks    1437 inodes
```

This means that in the */dev/root* filesystem, there are 5956 blocks and 1437 inodes free. 5956 blocks is roughly equivalent to 3 megabytes.

When used with the **-v** option, **df** reports on the percent of blocks used as well as the number of blocks used and free. Enter the following command:

df -v

Your output should look like the following:

```
Mount Dir   Filesystem   blocks   used    free    % used
/           /dev/root     80152   70192    9960     88%
/y          /dev/y        82194   34314   47880     42%
/u          /dev/u        50000   37840   12160     76%
```

This output indicates that on the */dev/root* filesystem 88% of the blocks, or 70192 blocks out of a total of 80152, are used. 9960 blocks are still free.

Note

Depending on how your system administrator has set up your system, you may not be allowed to use the **df** command. See ''Using a Trusted System'' in the *User's Guide* for details.

Controlling Processes

A command that is executing is considered a *process*. On a UNIX system, a user can run several processes at the same time, one in the *foreground* and several others in the *background*. The foreground process is the one that is currently executing on your terminal. This is the only process that can accept input from your keyboard. For instance, when you are editing with **vi**, the **vi** program is running as a foreground process.

Keyboard input cannot be sent to background processes. It is often useful to execute processes that are time consuming or require no keyboard input in the background. Controlling foreground and background processes is the subject of this section.

Placing a Command in the Background

5

Normally, commands sent from the keyboard are executed in strict sequence. One command must finish executing before the next command can begin. However, if you place a command in the background, you can continue to enter commands in the foreground, even if the background command is not finished executing.

To place a command in the background, put an ampersand (&) at the end of the command line. For example, enter the following command to create, and then count, the characters in a large file. Note that this command line is two lines long. This is made possible by placing the backslash (\) on the command line before pressing ⟨Return⟩. The backslash tells the shell that the command line continues on the next line:

```
cat /etc/termcap /etc/termcap /etc/termcap > largefile; \
wc -c largefile > characters &
```

The output of the **wc** part of the command is redirected to *characters*. If you issued this command without redirecting the output, the output would print on your screen, no matter what else you might be doing. This can be very disruptive. Redirecting the output of a background command to a file is a simple way of avoiding such disruptions.

Use **cat** to display the contents of *characters*. When you are finished, use **rm** to remove both *largefile* and *characters*.

When commands are placed in the background, you cannot abort them by pressing the **INTERRUPT** key, as you can with foreground commands. You must use the **kill** command to abort a background process. This command is described in ''Killing a Process,'' below.

Delaying the Execution of a Command

In addition to putting commands in the background, you can delay command execution. This is done with the **at** command. This command allows you to set up a series of commands to be executed at a specified time in the future. Use of this command is controlled by the system administrator; you can use it only after he or she has enabled you to do so.

The **at** command accepts standard input. The simplest form of the command is:

```
at time day < file
```

Note

> Depending on how your system administrator has set up your system, you may not be allowed to use the **at** command. See ''Using a Trusted System'' in the *User's Guide* for details.

The *file* argument is the name of the file that contains the command or commands to be executed. The *time* argument is the time of day, in digits, followed by ''am'' or ''pm.'' One- and two-digit numbers are interpreted as hours, three- and four-digit numbers as hours and minutes. You cannot use *time* arguments of more than four digits. The *day* argument is optional. It is either a month name followed by a day number, or a day of the week. If no *day* is specified, the command is executed the next time the specified *time* occurs.

For example, suppose that you have a large file that you want to print, but you don't want to do this during work hours because it will monopolize the printer for a long time. You could use **at** to print the file late at night, when nobody is in the office. To do so, first use **vi** to create a file containing the print command. Call the file *printfile*. This file might contain the following line:

```
lp filename
```

The *filename* argument is the name of the large file that you want to print.

After you create *printfile*, enter the following command:

```
at 11pm wed < printfile
```

There is no need to place this command in the background. Once you enter it and press ⟨Return⟩, the UNIX prompt reappears. This causes the command in *printfile* to be executed at 11:00 p.m. on Wednesday.

Note that **at** is unaffected by logging out. To display a list of files that you have waiting to be processed with **at,** use the **at -l** command. This command also lists the following information:

- The file's ID number.
- The command invoking the file (**at** or **batch**).
- The date and time the file is processed.

To cancel an **at** command, first check the list of files you have waiting to be processed and note the file ID number. Then use the **at -r** command to remove the file or files from the list.

The **at -r** command has the form:

at -r *ID-number*

For example, the following command removes file number 504510300.a, canceling whatever commands were included in that file:

```
at -r 504510300.a
```

Note that a user can only cancel his or her own files.

The files */usr/lib/cron/at.allow* and */usr/lib/cron/at.deny* control who has access to the **at** command. On many systems, only the super user is allowed to use **at**. Contact your system administrator if you need to use the **at** command but are denied access to it.

Finding Out Which Processes are Running

The **ps** command stands for "processes status" and displays information about currently running processes. This information is crucial if you need to kill a background process.

To display information about commands that you currently have running, enter the following:

ps

Your output should look like the following:

```
  PID  TTY  TIME  COMMAND
   49  2a   0:28  sh
11267  2a   0:00  ps
```

The PID column gives a unique process identification number that can be used to kill a particular process. The TTY column shows the terminal with which the process is associated. The TIME column shows the cumulative execution time for the process. The COMMAND column shows the actual command that is executing.

Enter the following command to display information about all the processes running on the system:

ps -e

To find out about the processes running on a terminal other than the one you are using, use the **-t** option and specify the terminal number. For example, to find out what processes are associated with terminal 2c, enter:

ps -t2c

Note

If the system administrator has not given you the appropriate permissions, you may not be able to see the processes of other users.

Killing a Process

To stop execution of a foreground process, press your terminal's **INTERRUPT** key. This is often the ⟨DEL⟩ key. Pressing this key will kill whatever foreground process is currently running. To kill all your processes executing in the background, enter the following command:

```
kill 0
```

To kill only a specified process executing in the background, you need to enter the following command:

kill *signal_number process_ID_number*

The *signal_number* is optional. It is sometimes needed to kill "stubborn" processes. Even stubborn processes can usually be killed with *signal_number* 9. Find out the *process_ID_number* with the **ps** command.

As an example, try killing the process associated with your shell. Note that when you log in to the system, you are placed in a shell. If you kill this shell process, you log yourself out. Enter **ps** and look at the process ID associated with either **sh** or **csh** in the COMMAND column. Suppose that this number is 4831. To kill your shell, enter the following command:

```
kill -9 4831
```

After entering this command, you see the login prompt again. Try it!

To guard against other users having control over your account, you can only kill your own processes.

Note

Killing a process associated with the **vi** editor can result in unpredictable terminal behavior. Also, temporary files that are normally created when a command starts, and then deleted when the command finishes, can be left behind after a **kill** command. Temporary files are normally kept in the directory */tmp*. You should check this directory periodically and delete your old files.

Shell Programming

Both the Bourne shell and the C-shell offer powerful programming features. If you have ever done batch programming in MS-DOS, you have some idea of what shell programming on a UNIX system is like. This section discusses the rudiments of shell programming. For a more complete discussion, see "The Shell" and "The C-Shell," both in the *User's Guide*.

In "Placing a Command in the Background," you were instructed to enter the following command:

```
cat /etc/termcap /etc/termcap /etc/termcap > largefile; \
wc -c largefile > characters &
```

However, you could have placed these commands in a file and executed the file.

Try it by creating a file called *command.file* with **vi**. Place the following lines in *command.file*:

```
cat /etc/termcap > largefile
cat /etc/termcap >> largefile
cat /etc/termcap >> largefile
wc -c largefile > characters
```

After placing these four lines in *command.file*, type **:x** to save it and quit **vi**. Now you must enter the following command to make *command.file* executable:

```
chmod +x command.file
```

Finally, execute *command.file* in the background by entering the following command:

```
command.file &
```

After a few moments, enter the following command to verify that *command.file* executed correctly:

```
cat characters
```

In fact, using some of the more sophisticated shell programming features that let you control the flow of a program, you could have written *command.file* as follows:

```
for name in /etc/termcap
do
      cat $name $name $name > largefile
done
wc -c largefile > characters
```

Whenever you place UNIX commands in a file, always remember to use the **chmod** command to make the file executable. (The **chmod** command is discussed in ''Using File and Directory Permissions'' in Chapter 4 of this tutorial.)

Summary

With UNIX systems, files are automatically stored in the computer's memory. There are provisions for creating backup copies of files, which can be stored on cartridge tapes or floppy diskettes. Once a backup copy has been made, you can display its contents, extract files from it, and run off additional copies. The operating system accepts both 3.5 inch and 5.25 inch diskettes that have any of the commonly-produced memory capacities (360K, 720K, 1.2M, 1.4M).

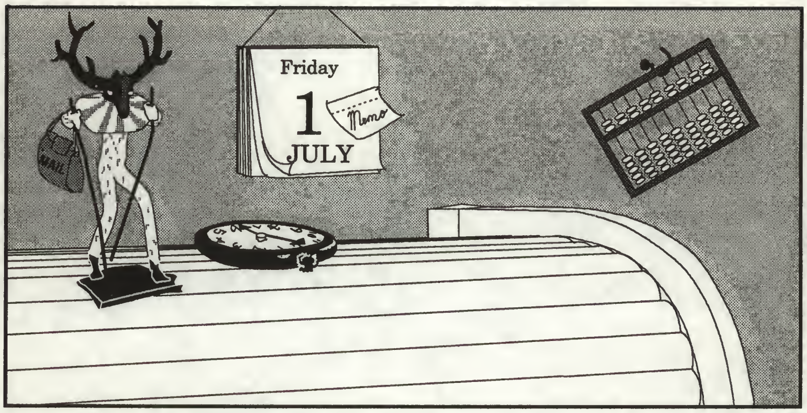
A currently-running command is called a *process*. On UNIX systems, you can run several processes at the same time by putting one in the *foreground* and the rest in the *background*. The foreground process is the one that shows up on your terminal screen, and is the only one that will accept input from your keyboard. When you enter commands, you must specify which ones should be put into the background. If you do not, the operating system assumes that they should all be in the foreground, and will execute them one at a time.

There are several commands for controlling the execution of processes. The commands covered in this chapter allow you to delay the execution of a process until a specified time, and to kill a foreground or background process.

There are also commands that provide information about the operating system itself. The commands presented in this chapter allow you to find out who is logged into the system, how much space is left on the system's main storage disk, and what processes are currently running.

Both the C-shell and the Bourne shell provide environments for writing very useful programs. You can write a *shell program* by creating a text file that contains a series of UNIX commands and then designating the file as executable. After that, the file (program) is executed whenever you enter the filename at the shell prompt.

For details about the commands presented in this chapter, see the *User's Guide* and the *User's Reference*.



Chapter 6

UNIX Desktop Utilities

Introduction

UNIX systems include a series of ''desktop'' utilities, programs that help you organize your work environment, and programs that allow you to communicate with other users on the system. This chapter describes these utilities, explaining how you can:

- Display the date, time and a calendar,
- Communicate with other users on the system,
- Use the system's automatic reminder service,
- Use the system's interactive calculator.

This chapter is designed as a tutorial. The best way to use this chapter is to read it at your terminal, entering commands as instructed in the examples.

None of the commands described in this chapter is described in great detail. For a complete explanation of each command, refer to the *User's Reference*.

Using the System Clock and Calendar

There are commands that display the date and time, as well as commands that display a calendar for virtually any month or year that you choose. The following sections explain these commands.

Finding Out the Date and Time

The **date** command displays the date and time. Enter:

date

Your output should look like the following:

```
Mon Jan 25 08:26:13 PST 1988
```

Displaying a Calendar

The **cal** command displays the calendar of any month or year that you specify. For example, to display the calendar for March 1952, enter:

cal mar 1952

The result is:

```
     March 1952

 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

The most common month abbreviations are accepted. The month can also be expressed as a digit. To display the calendar for an entire year, leave out the month. The year must always be expressed in full. The command **cal 88** displays the calendar for the year 88, not 1988.

Using the Mail Service

Several programs allow you to communicate with other users on a UNIX system. Two of the most useful are **mail** and **write**. The **mail** program allows you to send a message to a user's system "mailbox." The **write** program allows you to write a message directly to a user's terminal, if the user is logged into the system. Both of these programs are described below.

Sending Mail

The **mail** program is a system-wide facility that permits you to exchange mail with other users. Experiment with **mail** by sending a message to yourself. To do so, enter the following command:

mail *login*

Be sure to replace *login* with your login name.

Depending on how your system administrator has configured your mail system, you may see the following prompt:

```
Subject:
```

6

If you see this prompt, enter a short description of the message to follow. In this case, enter **test**.

You can now enter your message. When you are finished, press ⟨CTL⟩d to terminate message entry and mail your message. However, an alternative method is available for composing a message. If you enter **~v**, (a tilde, followed by a **v**) you are placed in **vi**. Once in **vi**, you can compose your message, using all the features available in **vi**. This method of composing a message is much more flexible than the other, as it allows you to correct mistakes in your message before you send it. Correcting mistakes in a message produced only with **mail** often results in control characters cluttering up the message.

If you use **vi** to compose your message, enter **:x** when you are finished, then ⟨CTL⟩d to terminate message entry. Again, depending on how your mail system is configured, you may or may not see the following prompt:

```
Cc:
```

If you do see this prompt, enter the names of those users who should receive ''carbon copies'' of this message. It is often convenient to **Cc:** yourself. Since this is a test message to yourself, just press ⟨Return⟩. You are now returned to the UNIX prompt.

Often you will want to send a text file to other users on the system. You can use the UNIX redirection facility to do this. Suppose that the file you want to send is named *schedule*, that it is in the current directory, and that you want to send it to users Naomi and Bea. To do so, enter the following command at the UNIX prompt:

mail naomi bea < schedule

Sending files this way is fast because you do not have to enter **mail.**

The **mail** facility has many options. These options allow you to include already composed messages in a mailing, to enclose a message you are responding to in your reply, to create mail aliases to send messages to several users at once, and so forth. These options are discussed in detail in ''mail'' in the *User's Guide.*

Receiving Mail

When you log in, you may see the following the message:

```
you have mail
```

To read your mail, enter:

mail

A list of message headers is displayed, each with a number in front of it. The list should look like the following:

```
 N  3 jill@gremlin.UUCP      Thu Jun 22 19:51   10/231   Notice
 N  2 sam@gremlin.UUCP       Thu Jun 22 19:51   10/231   Meeting
>N  1 tom@gremlin.UUCP       Thu Jun 22 19:51   10/231   Invite
```

Reading from left to right, the headers tell you the message is new (N), who the message is from, the date and time it was sent, the number of lines and characters in the message, and the message's subject. (The ''>'' symbol indicates the current message selected.)

To read a message, simply enter the number of the message you want to read, then press ⟨Return⟩. For example, to read the message from sam, enter **2** and then press ⟨Return⟩. Read the message that you sent to yourself, as described in the previous section.

There are several things you can do with your **mail** messages after you read them. You can delete them, save them, and/or respond to them.

To delete a message, press **d**, the message number and then ⟨Return⟩. To save a message in *filename* in the current directory, press **s**, the message number, type *filename* and press ⟨Return⟩. If *filename* does not exist, **mail** creates it.

Two commands are available for responding to mail. These are the **r** and **R** commands. If you press **r**, followed by the number of a message, followed by ⟨Return⟩, followed by a response, your response is sent to the author of the message. If you press **R**, followed by the number of a message, followed by ⟨Return⟩, followed by a response, your response is sent to the author of the message plus all users who were on the **Cc:** list of the original message.

After reading a message, you might want to list your message headers again. Do so by entering **h** followed by ⟨Return⟩. If you have more messages than will display on the screen, enter **h+** followed by ⟨Return⟩. This will cause the next screen of message headers to display. To display the previous screen of message headers, enter **h-**.

Note that you can send mail from within the **mail** program. To send mail to a user named ''jill'' from within **mail**, simply enter the following command:

```
mail jill
```

Then follow the steps outlined above in ''Sending Mail'' to send a message to jill.

To quit **mail**, enter **q** followed by ⟨Return⟩.

Respond to the message that you sent yourself by doing the following:

1. Enter mail by typing **mail**.
2. Enter **r***number*, where *number* is the number of the message that you sent to yourself.
3. Press ⟨Return⟩.
4. Compose your response. Remember that you can enter **~v** to compose your response in **vi**.
5. Press ⟨CTL⟩d to terminate your response. If you compose your response in **vi**, you will have to press **:x** to leave **vi** and then ⟨CTL⟩d.
6. Press ⟨Return⟩ to send the response.
7. **mail** will announce the arrival of the new message.
8. You should see your response to the message you sent yourself. Press the number that corresponds to this response to view it.
9. When you are finished looking at your messages, press **q** to leave **mail**.

Writing to a Terminal

The **write** program allows you to send messages directly to another user's terminal. You can reach another user as long as he or she is logged into the system and has not turned off write access to his or her terminal. For example, to write to joe's terminal, enter:

```
write joe
```

After you execute this command by pressing ⟨Return⟩, joe sees a message like the following on his terminal:

```
Message from login tty012...
```

The *login* in this message is your login name. To respond, joe enters:

write *login*

Again, *login* is replaced by your login name.

From this point on, each line that you enter is displayed both on your own terminal screen and on joe's. Each line that joe enters is displayed on both his screen and yours. To terminate the writing of text to joe, enter ⟨CTL⟩d alone on a line. Joe has to do the same to terminate his **write** session with you.

A typical procedure for coordinating communication in a two-way **write** is for each party to end each line with a distinctive signal, normally (o) for "over." The last line of a message is often followed by (oo) for "over and out."

Experiment with **write** by sending a message to yourself. Do so by entering the following command:

write *login*

Replace *login* with your login name. You should see a message like the following:

```
Message from login ttynn...
```

Now simply enter your message. Since you are writing to yourself, everything you enter appears on *your* screen twice.

For example, your **write** session might look like the following:

```
Hello Mark o
Hello Mark o
Remember, we have a meeting at 12:00. o
Remember, we have a meeting at 12:00. o
Right, see you there. oo
Right, see you there. oo
```

Press ⟨CTL⟩d to terminate the **write** session.

If you do not want to be interrupted by a message, enter the command **mesg n** any time after you have logged in. Anyone that tries to write to you after you have entered this command is denied permission. If you want to once again be accessible via the **write** program, enter **mesg y**. If you want write access to automatically be turned off whenever you log in, put the **mesg n** command in your *.login* file.

Using the Automatic Reminder Service

An automatic reminder service is available to all users. Once each day, the operating system automatically searches each user's home directory for a file named *calendar,* the contents of which might look like the following:

```
1/23 David's wedding
2/9  Mira's birthday
3/30 Paul's birthday
4/27 Meeting at 2:00
9/1  Karen's birthday
10/3 License renewal
```

Each line of *calendar* is examined. Lines containing today's and tomorrow's dates are extracted and mailed to you. To look at these reminders, you must invoke **mail**.

The file *calendar* is not created for you automatically. You have to create it yourself if you want to use this reminder service. It must be in your home directory.

Use **vi** or any other UNIX text editor to create and edit *calendar*. Be sure to place each date/event entry on a separate line. Dates can be specified in a variety of formats. Any of the following is acceptable:

9/7
Sep. 7
september 7

Using the Calculator

The **bc** command invokes an interactive desktop calculator that can be used like a hand-held calculator. A typical session with **bc** is shown below. Note that the session is begun by entering **bc** at the UNIX prompt and ended by entering **quit** on a line by itself. Comments explain what action is performed after each input line.

Action	Comment
bc	Activate bc
123.456789 + 987.654321	Add and output
1111.111110	
9.0000000 - 9.0000001	Subtract and output
-.0000001	
64/8	Divide and output
8	
1.12345678934 * 2.3	Note precision
2.58395061548	
19%4	Find remainder
3	
3^4	Exponentiation
81	
2/1*2	Note precedence
4	
2/(1*2)	Note precedence again
1	
x = 46.5	Assign value to x
y = 52.5	Assign value to y
x + y + 1.0000	Add and output
100.0000	
obase=16	Set hex output base
15	Convert to hex
F	
16	Convert to hex
10	
64	Convert to hex
40	
255	Convert to hex
FF	
256	Convert to hex
100	
512	Convert to hex
200	
quit	Must type whole word

Also available are scaling, function definition, and programming statements much like those in the C programming language. Other features include assignment to named registers and subroutine calling. For more information, see ''bc: A Calculator,'' in the *User's Guide*.

Summary

UNIX systems have several utilities that help you organize your work environment and communicate with other users on the system. There are UNIX utilities for:

- Displaying the current date and time,
- Displaying a calendar for any month of any year,
- Reminding you of upcoming events,
- Invoking an interactive calculator.

UNIX systems include a electronic *mail* system that lets you send and receive messages to and from other users. When you send mail to someone, your message is stored in the recipient's *mailbox* until he or she retrieves it. By using this format, the mail utility does not interrupt other users, and allows messages to be sent to users who are not currently logged in.

In addition to the **mail** utility, there is a *write* utility that lets you communicate directly with another user. To use this utility, both the sender and the recipient must be logged in. As each person types a message, it immediately appears on both users' terminals.

For details about the commands and utilities presented in this chapter, see the *User's Guide* and the *User's Reference*.

Index

5-22-89
512-210-960

512-210-960